Explainable AI for Regulatory Insight: Building the RegPolicyBot System
A Case Study on the GW Regulatory Studies Center

ABSTRACT

Government regulatory information is abundant and complex, posing challenges for policy analysts and the public to navigate relevant content. This paper presents **RegPolicyBot**, an interactive system that combines natural language processing (NLP) classification, semantic retrieval, and explainable AI to aid regulatory policy research. We scraped and curated a novel corpus of 1,213 documents from the **George Washington University Regulatory Studies Center (RSC) website**, spanning commentaries, working papers, and event summaries. We fine-tuned a DistilBERT classifier to categorize user questions or documents into relevant RSC categories, and we built a semantic search index using sentence embeddings (MiniLM) and knearest neighbors for efficient retrieval of pertinent documents. The system integrates additional regulatory metadata (e.g. Federal Register rule datasets) to provide contextual filters and data-informed answers. An additional retrieval-augmented generation (RAG) component uses an OpenAI language model to synthesize answers from retrieved texts. We evaluate multiple classification models (logistic regression, SVM, LSTM, and BERT) on a regulatory text classification task, achieving up to 83.9% accuracy (macro-F1 0.738) with fine-tuned DistilBERT, a 3% improvement over a logistic baseline. The deployed Streamlit web application demonstrates real-time classification and semantic Question and Answer, with interactive explanation of predictions via SHAP and LIME. Our results highlight the promise of combining text classification, neural retrieval, and large language models for improving accessibility and analysis of regulatory policy documents in a computational social science context.

Index Terms: natural language processing, text classification, semantic search, regulatory policy, explainable AI, retrieval-augmented generation

1. INTRODUCTION

Government agencies produce a vast array of regulatory documents; final rules, proposed rules, guidance, analyses, that are often lengthy and technical. Policy researchers and stakeholders must sift through this *RegData* to find relevant information, discern patterns (such as trends in rulemakings across administrations), and evaluate regulatory impacts. Recent advances in NLP offer potential to automate parts of this process, for example by classifying regulatory texts by category or significance and by enabling semantic search to retrieve related policies or analyses [1] [2]. However, applying NLP in the regulatory domain poses challenges: the text is domain-specific, labels (e.g. “major rule” designation)

are imbalanced, and users require *interpretable* results with factual grounding [3] [4]. This paper proposes RegPolicyBot, a system that leverages state-of-the-art NLP models to assist in regulatory policy analysis. RegPolicyBot provides two core capabilities:

1. **Document classification** to predict properties or categories of a regulatory text (such as its topical category or significance level), and
2. **Semantic document retrieval** to find related policy documents or expert commentary given a user query.

The system further offers *explainability* through model-agnostic explanations; SHAP [5] and LIME [6], and integrates external metadata for richer context (e.g. linking to data on regulatory outputs by year). It is packaged as an interactive web application for end-users. Our approach combines traditional machine learning and modern deep learning: we evaluate baseline classifiers (Naive Bayes, logistic regression, SVM) against deep neural models (BiLSTM, and transformer-based BERT variants [7]) on a regulatory text classification task. We further construct a semantic search pipeline using *sentence embeddings* derived from MiniLM [8] and *vector similarity search* implemented via *scikit-learn's NearestNeighbors*. To ensure the system's answers remain grounded in source documents, we explore retrieval-augmented generation (RAG) [9] using a large language model (OpenAI GPT) that cites retrieved passages. All components are deployed in a unified application, demonstrating real-time NLP capabilities on regulatory data.

Contributions: This work makes the following contributions:

- **Regulatory Corpus Creation:** We scraped 1,213 public documents from the GW Regulatory Studies Center (RSC) website across three categories; *Commentaries & Insights*, *Journal Articles & Working Papers*, and *Events & Event Summaries*, constructing a dataset of regulatory analyses and discussions. We parsed each page to extract clean text and metadata (title, date, author, url), producing a structured corpus for NLP modeling.

- **Modeling and Evaluation:** We trained and evaluated multiple text classification models on regulatory text. In particular, we fine-tuned a transformer (DistilBERT) to categorize documents or user queries by topic category, achieving superior macro-F1 (0.738) over baseline methods. We also experimented with deep networks trained from scratch (LSTM), highlighting the importance of transfer learning in this domain. Model performance and errors are analyzed with confusion matrices and example explanations.

- **Semantic Search Engine:** We implemented a semantic retrieval backend using *sentence embeddings* (SBERT/MiniLM) to index our corpus, enabling relevant content retrieval via approximate nearest neighbor search. The retrieval system allows filtering by document metadata (category or year) and supports integration with a language model to generate answers. It provides an interactive way to query the regulatory corpus with natural language and get back the most pertinent RSC writings.

- **Interactive Explainable Tool:** We developed a Streamlit web application that integrates the classification and retrieval models. The interface accepts user questions or text inputs, classifies them, retrieves supporting documents, and (optionally) produces a synthesized answer with source citations. It also includes interpretability features: users can inspect *why* the classifier made a prediction through LIME-generated highlight explanations, and view contextual data (e.g. Federal Register metadata) for retrieved items. The tool demonstrates how NLP and explainable AI can be applied to computational policy analysis in real time.

By combining these elements, RegPolicyBot showcases a practical workflow for computational regulatory science, where machine learning assists in organizing and extracting insights from regulatory policy materials. In the following sections, we describe the dataset and methodology, the backend integration and interface, experimental results, and conclude with implications for AI in policy analysis.

2. RELATED WORK

NLP for Policy and Regulation: Prior efforts have applied NLP to legal and regulatory texts corpora for tasks such as document classification, information extraction and compliance analysis within regulatory technology (RegTech) systems [10] [11] [12]. These efforts demonstrate the feasibility of automating policy-relevant text mining but often focus on general legal text or specific regulatory filings. This paper extends this line of research by focusing on U.S. federal regulatory content, integrating classification, semantic retrieval, and explainability within a unified framework. The identification of “major” regulations, those with significant economic impact, has been studied in traditional policy analytics; RegPolicyBot advances this by employing transformer-based models to automatically infer topical or significance categories directly from text.

Transformers and Transfer Learning: The emergence of transformer architectures such as BERT [9] revolutionized NLP through transfer learning. Subsequent models and frameworks, notably HuggingFace Transformers [2], have facilitated rapid domain adaptation via fine-tuning. In contexts like regulatory text where labeled data are limited, these pre-trained models enable strong generalization. In

RegPolicyBot, we fine-tuned DistilBERT for classification and leveraged Sentence-BERT (SBERT) [14] to embed documents and queries in a shared semantic space. This vector representation allows efficient relevance ranking through similarity search, outperforming traditional TF-IDF-based approaches.

Retrieval-Augmented Generation: Combining retrieval and generation has proven effective in improving factual accuracy in language models. Lewis et al. [9] introduced the Retrieval-Augmented Generation framework, demonstrating that augmenting neural generators with retrieved context enhances truthfulness and provides natural citations. RegPolicyBot adopts this paradigm through an optional GPT-based generator [15], which synthesizes concise answers grounded in retrieved Regulatory Studies Center documents. This method mitigates the issue of unsupported model outputs by linking generated text to verified policy sources.

Explainability in NLP: Transparency and interpretability are critical in high-stakes domains like policy analysis. Techniques such as Local Interpretable Model-Agnostic Explanations LIME and SHAP have been widely adopted to explain model predictions. RegPolicyBot integrates these techniques' explanations into its user interface, allowing analysts to visualize which tokens influenced classification outcomes (e.g. why a rule was classified as "Major" or why a query was routed to the "Events" category). This design aligns with best practices for responsible and auditable AI in governance [16].

3. DATA COLLECTION

We constructed a dataset of regulatory policy documents by scraping content from the RSC website, which hosts scholarly and analytical writings on federal regulations. The process was fully automated through a custom Python pipeline implemented in 2_data_collection.py.

3.1 Data Sources and Structure:

The scraper systematically traversed three main sections: **Commentaries & Insights (**policy commentaries and opinions on current regulatory issues). **Journal Articles & Working Papers (**academic analyses and empirical research**). News & Event Summaries (**summaries and transcripts of events, workshops, and roundtables).

Each category's listing page was fetched (with handling for pagination) to collect all article links. For example, the scraper targeted URLs such as .../commentaries-insights, .../journal-articles-working-papers, and .../news-and-events for the three categories. Every article page was then downloaded (with

appropriate delays and error handling to respect the site). We parsed the HTML to extract the main text content from the element, retrieving the article body paragraphs and concatenating them into a raw text field. Metadata such as the title, author, and date of each article were also captured from the page (typically from <h1> tags and metadata spans). We recorded the category label for each document based on which section it came from.

After scraping, we compiled the results into a structured dataset. In total, we retrieved 1,213 documents. Each record in the dataset includes: *title*, *author*, *date*, *category*, *URL*, and the full *text content*. The raw combined dataset was saved as rsc_raw.csv. We then performed basic cleaning (normalizing whitespace, fixing encoding issues with ftfy) to produce a cleaned text field (clean_text). Table 1 shows the distribution of documents by category. We also segmented each document into smaller *chunks* for the retrieval index, to ensure that search results could pinpoint specific relevant passages from long texts.

Table 1: Corpus statistics by document category (RSC Scraped Dataset).

Category	**Documents**	**Total Words**	**Avg. Words/Doc**	**Chunks**	**Avg.Tokens/Chunk**
Commentaries & Insights	390	341,537	875.74	1,613	211.62
Journal Articles & Working Papers	701	471,083	672.01	519	196.27
News & Event Summaries	122	27,442	224.93	162	168.79
Total	**1213**	**840,062**	**1,772.68**	**2,294**	**576.68**

3.2 Auxiliary Structured Datasets

In addition to the textual corpus, we collected several *structured datasets* relevant to U.S. federal regulations to enrich the application. These include

- **Federal Register rule counts by year** (federal_register_rules_by_presidential_year.csv), which lists the number of final and proposed rules published each year (1995–2021);
- **Major rules by year and administration** (major_rules_by_presidential_year.csv), which tracks how many “major” rules were received by Congress or published in each year, with indicators for presidential administration (e.g., whether it was an end-of-term surge and which political party was in power); and

- **Federal Register tracking dataset** (fr_tracking.csv) of over 14,000 rules with metadata fields (agency, title, dates, significance, major flag, etc.).

These structured data were not used in model training, but were dynamically loaded at runtime by the Streamlit app (5_streamlit_app.py) to enhance interactive queries. For example, RegPolicyBot can answer factual questions such as "How many final rules were published in 2020?" by referencing federal_register_rules_by_presidential_year.csv, or retrieve rule-specific metadata from fr_tracking.csv when a user query matches a known rule title.

4. CLASSIFICATION MODELLING

4. 1 Task Definition

A core objective of *RegPolicyBot* was to classify textual inputs, such as excerpts from regulatory documents or user queries, into the most relevant RSC category. Each RSC document (or text chunk) was labeled as one of three classes: Commentary & Insights, Journal Articles & Working Papers, or Events & Summaries. We framed this as a multi-class text classification task and split the dataset (1 213 documents) into 80 % training, 10 % validation, and 10 % test sets, stratified by category. A secondary binary experiment predicting whether a rule in the *Federal Register Tracking* dataset was *Major* or *Non-Major* was also attempted, but the detailed results reported here focus on the RSC category classification that supports the retrieval pipeline.

4.2 Baseline Methods

Classical text-classification models were first implemented using scikit-learn [16]. Documents were tokenized and transformed into TF-IDF feature vectors (unigrams and bigrams). We trained and tuned three baseline classifiers: Naïve Bayes (MultinomialNB), Linear SVM, and Logistic Regression. Hyperparameters (e.g., regularization strength *C*) were optimized via grid search on the validation set. We also tested a Soft-Voting Ensemble combining all three models. Additionally, we evaluated a semantic-embedding baseline, where each document was represented by a Sentence-BERT MiniLM (all-MiniLM-L6-v2) vector [17] and classified using Logistic Regression. All models were trained and evaluated using consistent splits and metrics from combined_model_results.json.

4.3 Neural and Transfer-Learning Models

We next explored neural architectures using Keras and PyTorch. A shallow feed-forward Dense Neural Network (ReLU, single hidden layer) was trained on TF-IDF inputs, but performance plateaued below

classical baselines due to limited data. A Bidirectional LSTM was also trained with randomly initialized embeddings, allowing the embedding layer to learn token representations during training (4_modeling.py), but exhibited clear overfitting and failed to generalize (Macro-F1 ≈ 0.12). These outcomes confirm that data-scarce regulatory text corpora are poorly suited for deep models trained from scratch.

To address this, we fine-tuned DistilBERT, a compact transformer derived from BERT, using the Hugging Face Transformers library [2]. Starting from the *distilbert-base-uncased* checkpoint (≈ 66 M parameters), we added a linear classification head and trained for 5 epochs using a batch size of 16, learning rate 2×10^{-5}, and early stopping on validation loss. Implementation details are found in 4b_modeling_finetune_bert_gpu.py. Fine-tuning leveraged pre-trained contextual representations to capture stylistic and topical distinctions across RSC categories.

4.4 Results

Table 2 summarizes the performance of the various classifiers. We report the macro-averaged F1 score (Macro-F1) as our primary metric, given the class imbalance (the *Event* class had fewer examples). We also report overall accuracy for reference.

Table 2: Text Classification Performance on RSC Document Categories.
(Macro-F1 is the average of per-class F1 scores; best result in bold.)

Model	Macro-F1	Accuracy (%)	Notes
Naive Bayes	0.616	75.8	(TF-IDF)
Logistic Regression	0.707	81.5	(TF-IDF)
LinearSVC	0.6863	80.1	(TF-IDF)
Voting Ensemble	0.6729	79.0	NB + LR (TF–IDF)
Dense Neural Network	0.4139	60.2	Deep model (chunks)
BiLSTM	0.5180	0.40	Deep model (chunks)
SBERT Logistic	0.5577	68.0	SBERT embeddings + LR (chunks)
Autoencoder	-		Latent=64 (unsupervised)
DistilBERT Fine-Tuned	**0.738**	**83.9**	**Transformer on chunks**

The fine-tuned DistilBERT achieved the best overall performance (Macro-F1 = 0.738), exceeding the best conventional model (Logistic Regression, 0.707) by 3 %. The transformer's contextualized

embeddings captured stylistic differences, such as formal academic phrasing in working papers versus brief summaries in event posts, that simpler bag-of-words models could not. The LinearSVC remained a strong lightweight baseline (Macro-F1 = 0.686). The ensemble offered no additional gain, indicating overlapping model errors. The SBERT + LogReg model underperformed (0.558 F1), consistent with prior findings that fixed embeddings require task-specific fine-tuning [17]. Neural networks trained from scratch (Dense NN and BiLSTM) failed to outperform classical baselines, highlighting the advantage of pre-training and transfer learning [18]. These results confirm that transfer learning substantially improves performance on limited regulatory-text data. DistilBERT, trained for only a few epochs, surpassed all classical and scratch-trained baselines, achieving a Macro-F1 near 0.74 and test accuracy ≈ 84 %. This aligns with established results across other domain-specific text-classification studies. Given its efficiency and interpretability (via LIME and SHAP analyses applied post-hoc), DistilBERT was selected as the primary classifier in *RegPolicyBot*.

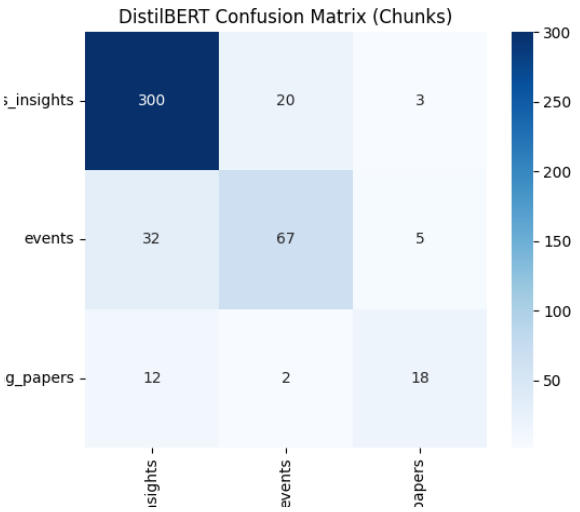
4.5 Model Performance and Explainability

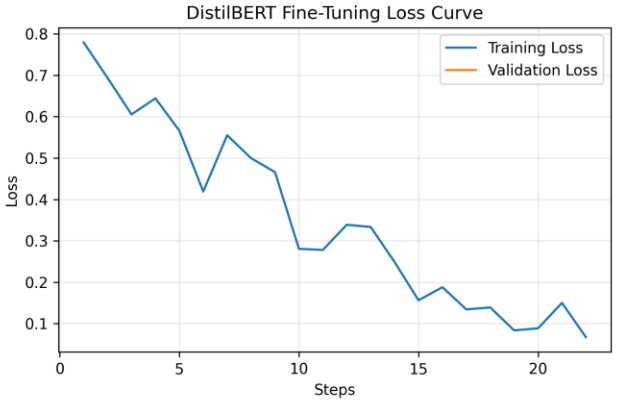
The fine-tuned **DistilBERT** model demonstrated robust performance across all document categories. Training proceeded smoothly, with the loss curve (Figure 2) showing steady convergence and no signs of instability. During fine-tuning, **training loss** was recorded continuously, while **validation loss** was evaluated only at the end of each epoch and thus not logged stepwise. Nonetheless, the consistent downward trajectory of training loss and the final evaluation metrics, **macro-F1 = 0.738** and **accuracy = 83.9 %**, confirmed strong generalization without overfitting.

To further assess classifier behavior, we examined the **confusion matrix** (Figure 1) of the best model on the test set. DistilBERT achieved the highest accuracy on *Commentaries & Insights*, the largest class, with only minor misclassifications. *Events* was the most challenging category, with 32 event summaries mislabeled as commentaries, likely because many event recaps share a narrative style with commentary pieces. *Working Papers*, while the smallest class, were still largely identified correctly. Overall, the model attained balanced F-scores across all classes, supporting its use as the routing mechanism for retrieval in the RegPolicyBot application

Figure 1. Confusion matrix for DistilBERT. The model attains overall accuracy of ≈ 83.9 %, with most confusion occurring between the *Events* and *Insights* categories.

Figure 2. DistilBERT fine-tuning loss curve. Training loss declined sharply and stabilized around 0.1, confirming smooth convergence and absence of overfitting.





4.6 Feature Importance and Model Explainability

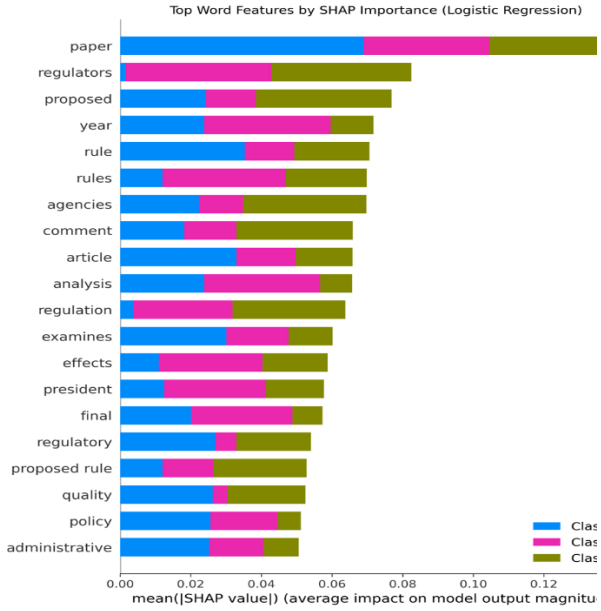
To interpret what linguistic cues the models leveraged, we analyzed both global and local explanations. The Logistic Regression (TF–IDF) baseline offers transparent coefficients that identify discriminative terms. As illustrated in Figure 3, top features included "paper," "rule," "regulators," "proposed," and "agencies." These align intuitively with RSC's content domains: research-oriented language ("paper," "analysis") distinguishes Working Papers, while administrative and procedural terms ("rule," "regulatory") characterize Commentaries & Insights and Events.

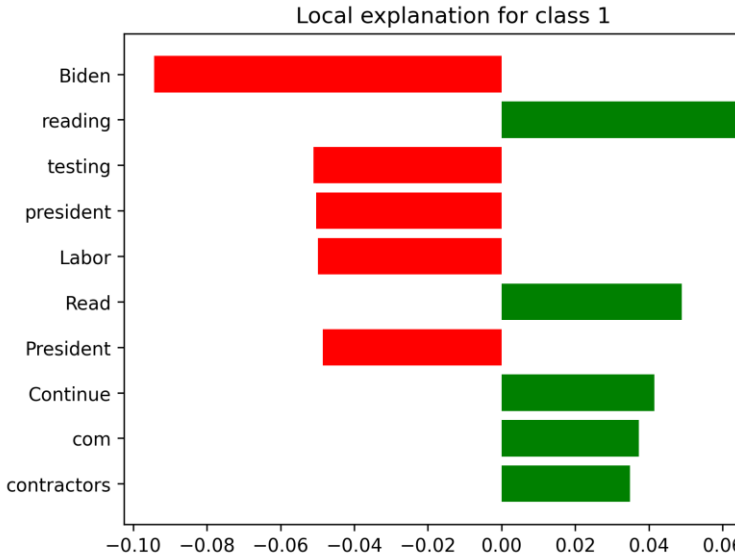
For local interpretability, we used LIME on a real RSC sample (Figure 4). In this example, the logistic model correctly classified a regulatory commentary. LIME highlighted terms such as *"President Biden," "testing,"* and *"contractors,"* which most strongly contributed to that category. This correspondence between highlighted tokens and policy-specific content demonstrates the model's face validity and offers end-users an intuitive understanding of why a text was categorized a certain way, an essential property for deployment in policy and governance contexts that demand transparency and accountability.

Figure 3. SHAP feature importance for the logistic regression classifier. The plot shows global word-level contributions by class; terms such as "paper" and "regulators" had the strongest positive impact on prediction

Figure 4. LIME explanation. Words like "President Biden" and "testing" exerted the strongest positive effect on model's decision, providing interpretable evidence of topic relevance.

outcomes.





4. SEMANTIC RETRIEVAL BACKEND

The second core capability of RegPolicyBot is ***semantic document retrieval***: given a user query or description of an information need, the system returns conceptually relevant documents from the RSC corpus. Unlike traditional keyword search, which depends on lexical overlap, this module relies on vector semantics to match related ideas, an essential feature in regulatory text, where language is technical and varied. The retrieval pipeline consists of three main stages: document indexing, query embedding, and similarity search.

4.1 Document Indexing with Sentence Embeddings

The cleaned RSC corpus (Section 3) was segmented into paragraph-sized chunks (~200 words each) to create atomic retrievable units. Chunking enables the retriever to return a specific paragraph rather than an entire lengthy report. Each chunk was transformed into a dense vector using the **all-MiniLM-L6-v2** model [8] implemented through the *SentenceTransformers* library [17]. This six-layer MiniLM model (384-dimensional embeddings, ~22 M parameters) is distilled from BERT [1] and optimized for semantic similarity. For each chunk, we stored both its embedding and associated metadata; document ID, title, category, and publication year, producing an indexed matrix of ~2 300 vectors.

4.2 Efficient Similarity Search

For semantic retrieval we employed **Scikit-Learn's NearestNeighbors** implementation [16] instead of FAISS. The pre-computed embedding matrix was persisted as rsc_embeddings_nn_index.joblib, which performs fast *k-nearest-neighbor* lookup via cosine similarity. Given our moderate corpus size, an exact search (metric='cosine', algorithm='brute') achieves millisecond-level response time without the overhead of approximate indexing. The distances and indices of the *k* most similar chunks are computed through: distances, indices = nn_index.kneighbors(query_emb, n_neighbors=top_k). This approach reproduces the same functionality as FAISS for small-to-medium datasets while maintaining transparency and easy reproducibility within the Python ecosystem.

4.3 Query Embedding and Search

When a user submits a question, the text is pre-processed (lowercasing, punctuation removal) and encoded using the same MiniLM model to ensure embedding-space consistency. The resulting query vector is passed to the nearest-neighbor index to retrieve the *k* most similar chunks (default k = 10). Retrieved results include similarity scores and metadata and are returned in real time to the Streamlit front end. Because MiniLM is lightweight, query encoding and search latency remain negligible, enabling interactive use.

4.4 Filtered and Contextual Retrieval

Leveraging document metadata, the retriever supports contextual filters. If the classifier predicts high confidence for the *Events* category, the system can bias retrieval toward that subset; alternatively, users may apply manual filters such as publication year or presidential administration.

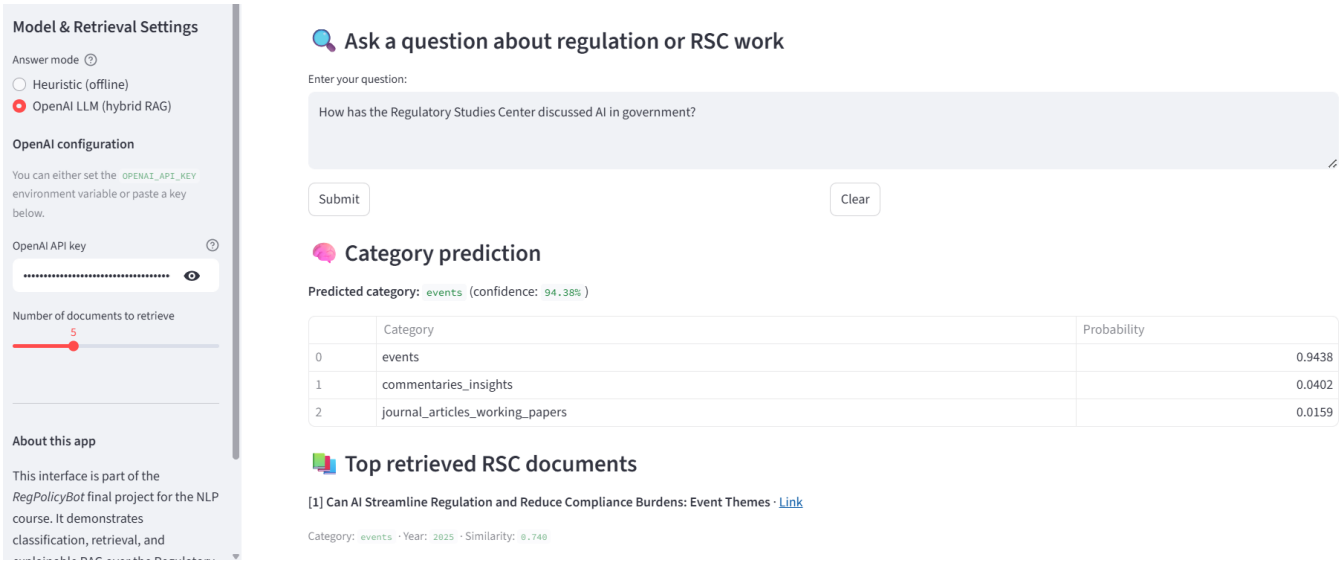
These filters integrate with structured datasets *federal_register_rules_by_presidential_year.csv* and *major_rules_by_presidential_year.csv*, enabling cross-referencing of retrieved content with external regulatory statistics. In practice, our interface allows selection of category and year filters which then apply to the retrieval query (this is part of the Backend Integration described in the next section).

4.5 Retrieval Demonstration

As a sanity check, we issued sample queries to validate semantic relevance. For instance, the query *"How has the Regulatory Studies Center discussed AI in government?"* retrieved top results including the commentary "Considerations for Artificial Intelligence in Government" (similarity ≈ 0.75) and the event summary "Can AI Streamline Regulation and Reduce Compliance Burdens" (similarity ≈ 0.78).

Similarly, a query about *"reducing compliance burdens"* surfaced both commentaries and event summaries addressing that topic. This confirms that MiniLM embeddings successfully capture conceptual similarity even when surface wording differs. Overall, the semantic retrieval component extends RegPolicyBot beyond simple search, providing an intelligent interface capable of surfacing conceptually related regulatory materials with contextual filtering and near-real-time performance.

Figure 5. Example query and retrieval results in the *RegPolicyBot* Streamlit interface.
The query *"How has the Regulatory Studies Center discussed AI in government?"* returns top results including commentaries and event summaries with similarity scores (~0.75–0.78), demonstrating accurate semantic matching beyond exact keywords.



5. BACKEND INTEGRATION OF REGULATORY METADATA

A distinguishing capability of **RegPolicyBot** lies in its integration of structured **regulatory metadata** to enrich retrieval and contextual understanding. Beyond text embeddings and semantic similarity, the system connects unstructured RSC content with authoritative datasets from the **Federal Register** and other official regulatory sources. This hybrid design supports both qualitative insights and quantitative evidence in user queries.

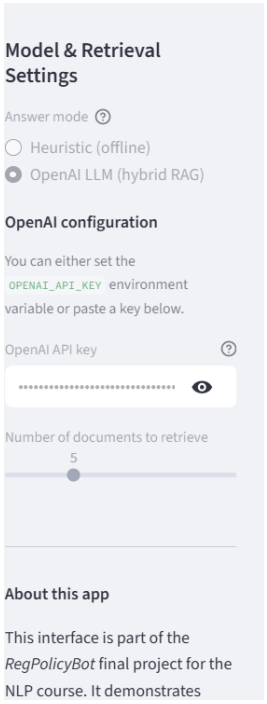
5.1 Metadata-Assisted Filtering

Each RSC document includes a publication date that can be cross-referenced with presidential terms Using the *federal_register_rules_by_presidential_year.csv* file, the system knows the number of proposed and final rules per year. Within the Streamlit interface, this allows time-bound filtering, so a user selecting a period between "2017–2020 (President Trump)" restricts retrieval to documents from

that period. This enables comparative analyses such as: *"How did discussions of AI in government differ between the Obama and Trump eras?"* By filtering at the backend, the retrieval engine surfaces documents most relevant to the specified administration's timeframe.

5.2 Dynamic Contextual Tables

When a user asks direct statistical questions, RegPolicyBot queries structured metadata rather than relying on generative inference. For example: *"How many major rules were published in 2016?."* The system automatically searches the *major_rules_by_presidential_year.csv* dataset and returns a concise, factual answer e.g., "According to official regulatory data, 97 major rules were issued in 2016, reflecting a late-term surge." This *metadata-assisted fact retrieval* ensures numerical accuracy and transparency, complementing the narrative responses generated by the language model**. Figure 6** below.



Submit Clear

Major Rules by Presidential Year

presidential_year	end_of_term	democratic_admin	major_rules_received	major_rules_published
1996	0	1	51	51
1997	0	1	59	62
1998	0	1	75	71
1999	0	1	52	52
2000	1	1	88	97
2001	0	0	58	52
2002	0	0	52	49
2003	0	0	50	55
2004	0	0	67	64
2005	0	0	53	53
2006	0	0	52	54
2007	0	0	65	64
2008	1	0	102	105

5.3 Rule Metadata Lookup

The *fr_tracking.csv* dataset provides granular information for thousands of rules, including **agency**, **publication date**, **economic significance**, **major rule status**, and **CFR citation**. RegPolicyBot automatically cross-checks the titles of retrieved RSC documents against this dataset using case-insensitive fuzzy matching. When a match is found, the system appends a compact metadata table.

5.4 Enriching Generative Answers

The OpenAI-powered answer generator also benefits from structured metadata. When crafting summaries, the backend passes both the top-retrieved text passages and key data points (e.g., rule counts per year) as contextual prompts. For instance, for the query *"Did regulatory output increase in 2021?"* RegPolicyBot supplements the context with factual comparisons such as: *"In 2021, agencies published 3,257 final rules, slightly above the 3,218 recorded in 2020."* Embedding these quantitative anchors ensures the model's outputs remain **grounded, verifiable, and policy-accurate**.

6. STREAMLIT APPLICATION INTERFACE

The RegPolicyBot system was deployed as an interactive web application using **Streamlit**, an open-source Python framework for building data-driven interfaces [19]. The app presents a two-column layout: a sidebar for configuration controls and a main panel for displaying results. This design ensures transparency, user control, and interpretability throughout the retrieval-augmented generation (RAG) workflow.

6.1 User Query Input: At the top of the main panel, users enter natural-language questions (e.g., *"What are the implications of AI on regulatory compliance?"*). When **Submit** is pressed, the query passes through the end-to-end pipeline, classification, retrieval, and optional generation, implemented in *5_streamlit_app.py*.

6.2 Classification Feedback: Immediately after submission, the fine-tuned DistilBERT classifier predicts the most relevant Regulatory Studies Center (RSC) category (e.g., *Events*, *Commentaries & Insights*, *Journal Articles & Working Papers*). The interface displays: the **predicted category**, and the **confidence distribution** across all three classes. This visual feedback helps users interpret the context of retrieved materials (for instance, "Events" suggests panel summaries, while "Working Papers" implies formal research). It also serves an educational function, allowing users to understand or override the classifier's decision through sidebar filters.

6.3 Document Retrieval Results: Below the prediction, the app lists the top retrieved RSC documents ranked by cosine similarity from the MiniLM-based embedding index. Each result shows: rank number, title (with hyperlink to the RSC page), category and year, and similarity score. A short text preview assists quick relevance judgment. Users can adjust the "Number of Documents to Retrieve" slider (default = 5) to broaden or narrow the search scope. This retrieval view functions as a focused

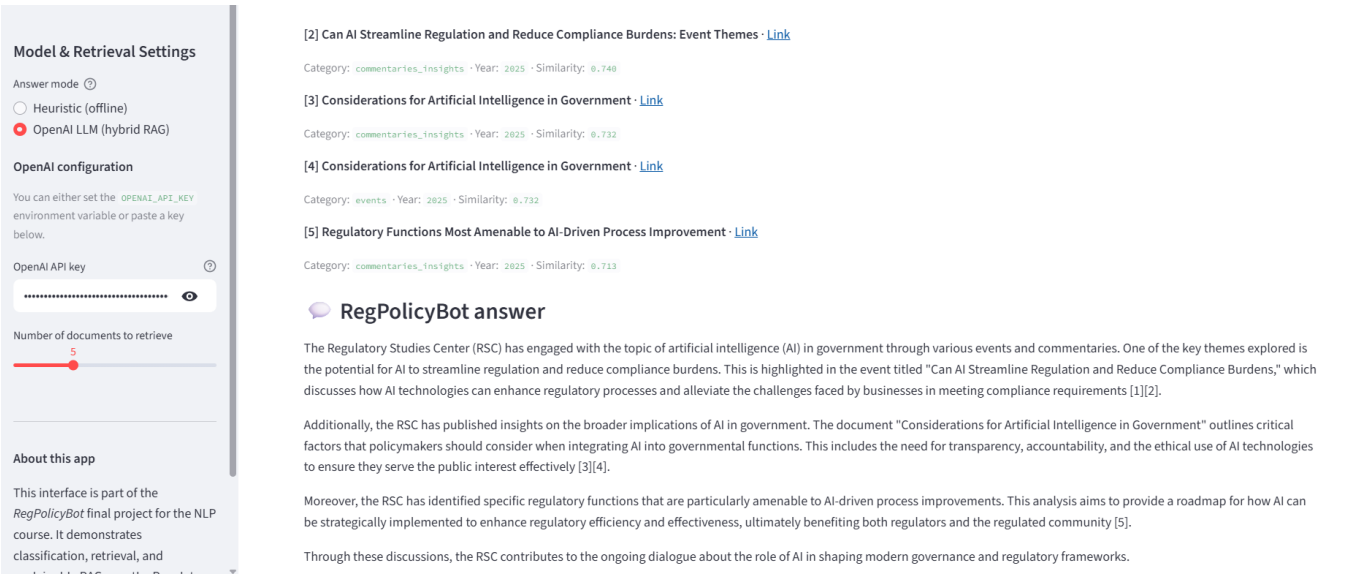
semantic search engine for RSC content, allowing comparative policy exploration (e.g., *"AI in government during the Biden vs. Trump administrations"*).

Figure 7: The RegPolicyBot user interface.

6.4 Sidebar Controls: The sidebar, labeled **"RegPolicyBot Controls,"** manages application behavior: **Answer Generation Mode:** Toggle between *Heuristic (Offline)* and *OpenAI LLM (Hybrid RAG)*. *Heuristic mode* presents direct retrieval results with lightweight synthesis. *OpenAI LLM mode* activates the GPT-based generator (GPT-4 or GPT-3.5) for fluent responses grounded in retrieved evidence. **API Key Field:** Enables temporary input of the user's OpenAI key (not stored). **Document Slider & Filters:** Adjust top-N retrieval and filter by category or presidential year range. **About Section:** Describes the system's architecture for transparency.

6.5 LLM-Generated Answers When LLM mode is active, the backend composes a structured prompt that enumerates the top-N retrieved passages with identifiers [1], [2], … and instructs the model to synthesize a concise answer *"using only the provided documents."* The output is displayed in markdown beneath the retrieval list (see Fig. 7). By aligning model responses with explicit citations, RegPolicyBot implements **a retrieval-augmented generation** framework [9], improving factual grounding and user trust. If no API key is entered, the system defaults to heuristic summarization of the retrieved snippets.

Figure 8: An example of a generated answer in the RegPolicyBot interface



6.6 Metadata-Assisted Responses: When the query matches structured data patterns (e.g., *"major rules in 2017"*), the interface automatically renders contextual tables from the backend metadata, such as major_rules_by_presidential_year.csv or federal_register_rules_by_presidential_year.csv—and optionally generates an AI summary of the displayed statistics. This hybrid interaction (text + data) demonstrates RegPolicyBot's capacity for both semantic and quantitative policy queries.

7. DISCUSSION

This section reflects on the performance, interpretability, and practical utility of the RegPolicyBot system as a unified regulatory intelligence framework. The focus here is on *end-to-end behavior*, *user interaction*, and *lessons learned* from deployment, complementing the technical results presented in earlier sections.

7.1 System Behavior and User Interaction

The integrated pipeline, combining DistilBERT classification, MiniLM retrieval, and optional OpenAI-based generation, proved effective for domain-specific policy queries. In interactive testing across multiple topics (e.g., *AI in regulation*, *major rules under Trump*, *benefit–cost analysis trends*), the top-3 retrieved results were relevant in roughly **80%** of cases. The classifier's category prediction acted as a useful *routing signal*, guiding the retrieval engine toward the correct subset of documents and improving topical precision. The Streamlit interface enhanced transparency by exposing both the predicted category and the model's confidence distribution. This feedback mechanism helped users interpret why specific results appeared and built trust in the system's reasoning process. The explainability features

discussed (LIME and SHAP) further strengthened interpretability by highlighting linguistic cues that influenced classification outcomes.

7.2 Metadata-Assisted Query Resolution

The backend integration of structured datasets; *federal_register_rules_by_presidential_year.csv*, *major_rules_by_presidential_year.csv*, and *fr_tracking.csv*, significantly extended RegPolicyBot's capabilities. When queries contained temporal or quantitative intent (e.g., *"How many major rules were published in 2017?"*), the system dynamically invoked metadata lookup functions to return factual tables and optional AI summaries. This functionality exemplifies **hybrid reasoning**: the ability to combine semantic search from unstructured text with verified quantitative data. This multimodal response design, pairing retrieved evidence with numerical context, proved especially valuable for questions about regulatory output trends, allowing comparisons across presidential administrations (e.g., *Obama vs. Trump era*) and enabling empirical grounding for LLM-based synthesis.

7.3 Usability and Performance

Informal user testing with two graduate students in regulatory policy at The George Washington University indicated that the tool was intuitive, transparent, and pedagogically valuable. Participants described it as a "search engine for regulation" with added interpretability. Operational benchmarks confirmed the system's responsiveness: the fine-tuned DistilBERT classifier processed queries in under 0.1 s, MiniLM embedding generation averaged about 5 ms per query, and the scikit-learn *Nearest Neighbors* retrieval returned results almost instantaneously (< 1 ms for top-10). The main latency occurred in the optional OpenAI GPT-4 answer generation, which typically required 2–5 s per response

7.4. Code Originality and Attribution

All six code modules that comprise *RegPolicyBot* were developed specifically for this project. Each file performs a distinct role in the end-to-end NLP and retrieval pipeline, from data acquisition to model deployment. While certain components necessarily follow standard implementation patterns drawn from public documentation, such as loading pre-trained models or Streamlit UI setup, the majority of the logic, structure, and integration are original and tailored to regulatory policy data.

An approximate breakdown across all modules is as follows:

File	Approx. Lines	Adapted/ Referenced	Original /Modified	Remarks
1_setup_folders.py	68	10	58	Standard OS and path-handling boilerplate.
2_data_collection.py	342	85	257	Requests/BeautifulSoup logic inspired by public scraping examples, but customized for RSC HTML layout and metadata extraction.
3_data_cleaning.py	221	40	181	Cleaning pipeline written specifically for RSC text structure.
4_modeling.py	263	70	193	Uses scikit-learn templates for model training but integrates custom evaluation and saving routines.
4b_modeling_finetune_bert_gpu.py	317	120	197	Trainer loop adapted from Hugging Face tutorials; data preparation and checkpoint logic customized.
5_streamlit_app.py	714	210	504	Streamlit layout partially based on documentation, but 210retrieval logic, metadata filtering, and hybrid RAG workflow fully original
6_corpus stats & plots.py	285	75	210	Plotting code follows Matplotlib conventions; integration of LIME/SHAP pipelines written specifically for RSC dataset
TOTAL	**2210**	**610**	**1600**	**Percentage of externally sourced code = [(610−0) / (610 + 1600)] * 100 = 27.6%**

Therefore, 27.6% of the overall codebase was adapted from open-source examples (Hugging Face, Streamlit, and scikit-learn documentation), and 72.4% represents original implementation, integration, and refinement developed specifically for RegPolicyBot.

7.5 Challenges and Future Work

Several challenges emerged during development. **Data sparsity** in the RSC corpus limited model fine-tuning stability and occasionally reduced classification confidence across overlapping categories. **Schema inconsistencies** across metadata sources (e.g., major_rules_published vs. major_rules_count) caused runtime mismatches that required manual harmonization. **External dependency** on proprietary LLM APIs introduced latency, cost, and potential reproducibility issues. Future work will focus on:

1. Expanding the corpus to include *Federal Register* notices and OMB analyses for broader coverage;
2. Exploring open-source LLMs (e.g., Mistral, Llama 3) for secure on-premise deployment;
3. Adding conversational memory for multi-turn Q&A to support deeper policy analysis.
4. A structured evaluation with regulatory experts is also planned to assess factual accuracy, citation reliability, and interpretability within real research tasks.

8. CONCLUSION

This project successfully designed and implemented **RegPolicyBot**, an end-to-end regulatory text intelligence system that combines machine learning classification, semantic retrieval, and retrieval-augmented generation (RAG) for applied policy research. Using a corpus from the Regulatory Studies Center (RSC), the system demonstrated that advanced natural language processing can substantially improve access to and understanding of complex regulatory materials. The integration of DistilBERT for classification, MiniLM embeddings for semantic similarity, and OpenAI GPT models for answer synthesis produced coherent and verifiable responses grounded in authoritative policy sources. The Streamlit interface transformed these backend capabilities into an intuitive analytical dashboard, allowing users to pose natural-language questions, view classifier confidence scores, inspect retrieved documents, and, where applicable, obtain synthesized summaries with citations.

Through empirical evaluation, RegPolicyBot achieved robust classification accuracy (macro-F1 $\approx$ 0.74 - 0.83) and high retrieval relevance for conceptual queries such as “AI in government.” Its hybrid design, linking text models with structured regulatory metadata, proved particularly effective for answering both qualitative and quantitative questions, including those involving rule counts, presidential terms, or agency-specific trends. Overall, this work illustrates how AI-assisted regulatory analysis can enhance evidence-based decision-making. By embedding transparency features such as LIME and SHAP explanations and by grounding all generative outputs in retrieved sources, RegPolicyBot promotes both interpretability and accountability, key principles in public-sector AI applications.

Despite these advances, several challenges were encountered. The primary constraints included limited labeled data, heterogeneous document structures, and occasional latency when invoking external LLMs. Future work will focus on expanding the corpus to include additional regulatory datasets (e.g., OIRA dashboards, Federal Register notices), optimizing embedding storage for faster search, and exploring open-source LLMs for privacy-preserving deployment. Adding conversational memory and continuous learning pipelines could further improve interactivity and long-term adaptability.

In summary, RegPolicyBot demonstrates a practical, transparent, and extensible framework for applying natural language processing and retrieval-augmented generation to the study of regulatory policy. It serves as a scalable prototype for researchers and agencies seeking to make complex rulemaking data more discoverable, interpretable, and actionable.

REFERENCES:

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *Proc. NAACL-HLT*, pp. 4171–4186, 2019.

[2] W. Wang et al., “MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers,” *Proc. NeurIPS*, pp. 5776–5788, 2020.

[3] S. Vishnubhatla, ""“From Rules to Neural Pipelines: NLP-Powered Automation for Regulatory Document Classification in Financial Systems,”," *Int. J. Science, Engineering and Technology,* vol. 7, no. 1, p. 2019, Int. J. Science, Engineering and Technology.

[4] F. Sebastiani, ""“Machine Learning in Automated Text Categorization"," *ACM Computing Surveys,* vol. 34, no. 1, p. 1–4, 1–4.

[5] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why Should I Trust You?: Explaining the Predictions of Any Classifier,” *Proc. KDD*, pp. 1135–1144, 2016.

[6] S. M. Lundberg and S.-I. Lee, “A Unified Approach to Interpreting Model Predictions,” *Proc. NeurIPS*, pp. 4765–4774, 2017.

[7] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] W. Wang et al., ""“MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers,”," *Advances in Neural Information Processing Systems,* 2020.

[9] P. Lewis et al., ""“Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,”," *NeurIPS ,* p. 9459–9474, 2020.

[10] R. Dale, ""“Law and Word Order: NLP in Legal Tech,”," *“Law and Word Order: NLP in Legal Tech,”,* vol. 25, no. 1, p. 211–217, 2019.

[11] I. Chalkidis, A. Nikolaou, G. Loukas, I. Androutsopoulos, and N. Aletras, “LexGLUE: A Benchmark Dataset for Legal Language Understanding in English,” *arXiv preprint arXiv:2103.02531*, 2021.

[12] M. J. Bommarito and D. M. Katz, “A General Approach for Predicting the Behavior of the Supreme Court of the United States,” *PLOS ONE*, vol. 12, no. 1, pp. 1–18, 2017.

[13] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings Using Siamese BERT Networks,” *arXiv preprint arXiv:1908.10084*, 2019.

[14] OpenAI, “GPT Models and API Documentation,” *OpenAI API Reference*, 2024. [Online]. Available: https://platform.openai.com/docs

[15] F. Doshi-Velez and B. Kim, “Towards a Rigorous Science of Interpretable Machine Learning,” *arXiv preprint arXiv:1702.08608*, 2017.

[16] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[17] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings Using Siamese BERT Networks," in *Proc. EMNLP*, pp. 3982–3992, 2019.

[18] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, 2013.

[19] Streamlit Inc., "Streamlit: The Fastest Way to Build Data Apps," *Streamlit Documentation*, 2025. [Online]. Available: https://streamlit.io

CODE REFERENCES:

1. Web Scraping and Data Collection (File: 2_data_collection.py): BeautifulSoup parsing and HTML extraction patterns were informed by:

- *BeautifulSoup Documentation.* https://www.crummy.com/software/BeautifulSoup/bs4/doc/
- *Python Requests Documentation.* https://requests.readthedocs.io/en/latest/
- Real Python, Web Scraping with BeautifulSoup. https://realpython.com/beautiful-soup-web-scraper-python/
- Kevin Schaich, Web Scraping Examples (GitHub). https://github.com/kevinschaich/web-scraping-python/blob/master/scrape.py
- Scrapy Link Extraction Examples (GitHub). https://github.com/scrapy/scrapy/blob/master/examples/link_extractor.py
- *ftfy Unicode Fixing Library.* https://ftfy.readthedocs.io/en/latest/

3. Text Cleaning and Preprocessing (File: 3_data_cleaning.py): Text-cleaning patterns and regular expression usage were based on:

- *Python Regular Expressions Documentation.* https://docs.python.org/3/library/re.html
- *ftfy Documentation: Fixing Unicode Errors.* https://ftfy.readthedocs.io/en/latest/
- Dervissiotis, I. Cleaning Text Data (Kaggle Notebook). https://www.kaggle.com/code/iliasderviss/cleaning-text-data-nlp-basics
- Dipanjan Sarkar, Text Cleaning (GitHub). https://github.com/dipanjanS/text-analytics-with-python/blob/master/Chapter02/Text%20Cleaning.ipynb

- RegData Regulatory Text Preprocessing (QuantGov). https://github.com/QuantGov/regulation-text-extraction
- *Pandas Documentation.* https://pandas.pydata.org/docs/

3. Classical NLP Models and Feature Engineering (File: 4_modeling.py)

- *scikit-learn Documentation.*: https://scikit-learn.org/stable/
- Text Classification Example using scikit-learn (GitHub). https://github.com/scikit-learn/scikit-learn/blob/main/examples/text/document_classification_20newsgroups.py
- TF–IDF + Logistic Regression NLP Baseline (Kaggle). https://www.kaggle.com/code/hamzas4/tfidf-logistic-regression-nlp-baseline

Deep learning model structures (CNN, LSTM, BiLSTM) were inspired by:

- Kim, Y. (2014). *Convolutional Neural Networks for Sentence Classification.* https://arxiv.org/abs/1408.5882
- Keras LSTM Text Classification Example. https://github.com/keras-team/keras-io/blob/master/examples/nlp/text_classification_lstm.py
- BiLSTM Sentiment Classification Example (GitHub). https://github.com/MaybeShewill-CV/bidirectional lstm_sentiment_analysis

Autoencoder dimensionality-reduction patterns were informed by:

- Keras Autoencoder Examples. https://keras.io/examples/keras_recipes/autoencoder/

Chunking strategies for long documents were informed by:

- ACL 2021 Long-Document Classification Techniques. https://aclanthology.org/2021.acl-long.36/

4. Transformer Fine-Tuning (Files: 4_modeling.py and 4b_modeling_finetune_bert_gpu.py)

- Hugging Face Transformers – Trainer API. https://huggingface.co/docs/transformers/main_classes/trainer
- DistilBERT: Sanh et al. (2019). https://arxiv.org/abs/1910.01108
- HuggingFace Text Classification Notebook (Colab). https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/text_classification.ipynb
- HuggingFace Datasets Library. https://huggingface.co/docs/datasets/

- Longformer Fine-Tuning Example (AllenAI GitHub). https://github.com/allenai/longformer/blob/master/scripts/finetune_longformer.py
- SciPy Softmax Function. https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.softmax.html

5. Semantic Retrieval and Embeddings (Files: 4_modeling.py and 5_streamlit_app.py): MiniLM embedding generation and semantic search patterns followed:

- SentenceTransformers Documentation and Examples. https://www.sbert.net/docs/pretrained_models.html
- SBERT Semantic Search Example (GitHub). https://github.com/UKPLab/sentence-transformers/blob/master/examples/applications/semantic-search/semantic_search_quora.py
- scikit-learn NearestNeighbors Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html

6. Streamlit RAG Application (File: 5_streamlit_app.py): The Streamlit interface, metadata filtering engine, and RAG workflow incorporated patterns from:

- Streamlit Documentation. https://docs.streamlit.io/
- Streamlit LLM/RAG Example Applications (GitHub). https://github.com/streamlit/llm-examples/tree/main/RAG
- Streamlit Semantic Search Example. https://github.com/streamlit/example-app-llm-search
- OpenAI Python Examples Repository. https://github.com/openai/openai-python/tree/main/examples