Hybrid Semantic Search Engine for Government RFP Discovery

Aaron Good
agood@oxy.edu
Occidental College

Abstract

Government procurement represents a $755 billion annual market where discovery still relies largely on keyword search, causing businesses to miss relevant opportunities due to vocabulary mismatch. This paper presents a hybrid semantic search engine for Request for Proposals (RFP) discovery that combines TF-IDF lexical matching with transformer-based semantic embeddings. We implement a working prototype that ingests federal opportunities from SAM.gov and fuses results using convex combination. Evaluation on a corpus of 55 RFPs demonstrates that keyword and semantic search retrieve substantially different document sets (29% overlap), and that hybrid search surfaces opportunities that either method alone would miss—on average, 51% of hybrid results had zero TF-IDF scores. Stakeholder interviews confirm that vocabulary mismatch and platform fragmentation are real barriers to procurement discovery. The system addresses vocabulary mismatch but highlights the need for unified procurement infrastructure aggregating federal, state, and municipal sources.

1 Introduction

Public procurement involves enormous volumes of information driving immense economic value: the U.S. federal government alone awarded roughly $755 billion in contracts in FY 2024 [3], and procurement contracts account for over 10% of the U.S. budget [5]. Each procurement opportunity (e.g., an RFP, or Request for Proposals) contains technical specifications and legal requirements, but these documents are often heterogeneous (PDFs, scanned notices, web pages) and spread across different private, federal, state, and municipal portals. Although platforms like the U.S. Federal Procurement Data System (FPDS) and leading enterprise product FindRFP.com provide keyword-based search interfaces [5], purely lexical search consistently fails to match relevant RFPs when different terminology or phrasing is used. For example, a query about "automated vehicle guidance" might miss opportunities described in terms of "driverless car navigation." This motivates a *hybrid* search approach combining classical keyword methods (such as TF-IDF or BM25) with semantic search using learned text embeddings, and more broadly in scope, a unified platform for government procurement intelligence.

Hybrid search seeks to leverage the precision of term-based matching and the robustness of meaning-based retrieval [2]. In procurement, minor wording differences can hide matches, so semantic search (using pre-trained transformer models like BERT) can reveal related results, while TF-IDF keyword-based approaches can ensure technical terms maintain concrete mechanism of relevancy. This paper presents a design for a hybrid semantic search engine, a prototype for such a unified platform, tailored to RFP discovery, aiming to improve recall and relevance over keyword-only systems. We discuss the motivation, technical foundations, related work (public procurement data and hybrid IR systems), system design (data pipeline, indexing, retrieval scoring), in addition to evaluation, broader ethical considerations, and considerations for future work. As governments and organizations increasingly move to open up procurement data (e.g., the World Bank's Global Public Procurement Database [12]), effective search tools become critical for equity in transparency and competition.

2 Technical Background

2.1 Term Frequency–Inverse Document Frequency (TF-IDF) and BM25

Classical information retrieval (IR) treats documents as bags of words and scores them by term weights. **TF-IDF** is a common weighting scheme built into most modern search functionality: it assigns each term t in document d a score proportional to its *term frequency* $tf(t,d)$ and inversely proportional to its frequency in the corpus. For example:

$$tfidf(t,d) = tf(t,d) \times \log \frac{N}{df(t)}, \tag{1}$$

where N is the total number of documents and $df(t)$ is the number of documents containing t [4]. Intuitively, this boosts terms that are frequent in one document but rare overall. TF-IDF (and variants thereof) have been used widely in search engines, text mining, and even search engine optimization [4]. A simple relevance score might sum the TF-IDF weights of all query terms in a document [4].

Okapi BM25 is a modern scoring function that extends TF-IDF with probabilistic reasoning and document-length normalization [7, 9]. In BM25, the score of document D for query terms $q_1, \ldots, q_n$ is:

$$\text{score}(D, Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})} \tag{2}$$

where $f(q_i, D)$ is the term frequency of q_i in D, $|D|$ is the document length (in words), and k_1, b are tunable parameters [9]. The inverse document frequency $IDF(q_i)$ further downweights common terms [9]. By default, Elasticsearch (a partially open-sourced search engine with semantic capabilities) and similar search engines use BM25 as a "Practical Scoring Function," combining TF-IDF and normalization for document length [1].

We use TF-IDF for the lexical retrieval component. While BM25 offers improved handling of document length variation, TF-IDF is sufficient for demonstrating hybrid retrieval, and the semantic component compensates for lexical limitations. TF-IDF ensures domain-specific terms (e.g., regulatory codes, NAICS classifications) are matched precisely if needed.

2.2 Semantic Search with Embeddings

In contrast to keyword search, **semantic search** represents text by continuous vector embeddings that capture meaning. A modern approach is to embed all documents into a high-dimensional vector space, and likewise embed each user query; then retrieval is done via nearest-neighbor lookup in this space [10]. In this setting, documents with similar *meaning* (even if vocabulary differs) have nearby embeddings, enabling discovery of relevant items via cosine similarity. For example, a query "renewable energy project" might retrieve an RFP labeled "solar and wind power plant," which uses different words but shares intent. The Sentence-Transformers library (a popular Python framework for creating high-quality, dense vector embeddings) explains: "Semantic search seeks to improve search accuracy by understanding the semantic meaning of the search query and the corpus... it can perform well given synonyms, abbreviations, and misspellings, unlike keyword search engines that can only find documents based on lexical matches" [10].

The core idea is to use a *sentence embedding* model to encode each document (or document segment) into a vector. At query time, the query is encoded and the nearest document vectors are returned, under the assumption that semantic similarity in embedding space reflects relevance [10].

Historically, large pre-trained transformer models like Google's BERT can produce powerful and highly accurate contextual embeddings, but require pairwise processing of query and document, which is impractical for large-scale search. The Sentence-BERT (SBERT) approach [8] uses a siamese BERT network to compute fixed-size sentence embeddings. Reimers and Gurevych (2019) showed that SBERT embeddings allow fast semantic search: "Finding the most similar pair in a collection of 10,000 sentences requires about 50 million inference computations ($\sim$65 hours) with BERT; our Sentence-BERT reduces this to about 5 seconds while maintaining accuracy from BERT" [8]. In practice, we adopt a similar approach, using the more lightweight SBERT all-MiniLM-L6-v2 to encode RFP texts. SentenceTransformers also provides an easy API to compute such embeddings [11]. Applied to RFPs in this project, we encode an RFP's title, summary, description, state, and NAICS codes into a 384-dimensional vector by concatenating relevant fields.

Semantic embeddings also support asymmetric tasks, where a short keyword query is matched against longer document embeddings. In this project, we compute document vectors once (offline) and at query time encode the user's query. Retrieval then finds nearest neighbors by cosine similarity (or dot product) in the vector index [10, 2]. The dense-vector search can capture broad topical matches beyond exact keywords.

2.3 Comparison and Combination of Lexical and Semantic Search

Our prototype engine leverages both independent TF-IDF and semantic mechanisms, as each have different strengths. A bag-of-words model like TF-IDF excels when queries use the same specific terms as documents (excellent *precision* for exact matches), but it fails to retrieve semantically related results if wording differs [7]. Semantic search, conversely, can find related content even under paraphrases, but may overlook important domain-specific keywords or be less precise for users looking for specific phrase matches. For example, an RFP explicitly requiring "mil-spec 810G certification" might be missed by a semantic model if the query doesn't mention "mil-spec," whereas a TF-IDF index would score it very highly. Thus, combining them can offer the best of both worlds: lexical search ensures critical terms are matched, while embeddings catch near-synonyms or related concepts.

Score Fusion. Given ranked results from both retrieval methods, hybrid systems must combine scores into a unified ranking. Common approaches include Reciprocal Rank Fusion (RRF), which sums inverse ranks, and convex combination, which computes a weighted sum of normalized scores:

$$\text{score} = \alpha \cdot s_{\text{tfidf}} + (1 - \alpha) \cdot s_{\text{semantic}}. \tag{3}$$

We adopt convex combination with $\alpha = 0.5$, giving equal weight to lexical and semantic signals. Scores are min-max normalized before combination to ensure equal contribution from each component.

3 Related Work

3.1 Government Procurement Data and RFP Access

Global initiatives reflect the demand for better procurement data access. The World Bank's Global Public Procurement Database (GPPD) aggregates country procurement indicators to promote transparency and best practices [12]. The U.S. maintains FPDS and the General Services Administration's SAM.gov, but these systems primarily support keyword queries and filtering by fields (agency, NAICS code, etc.) [5]. The Scientific Data publication by Omari et al. provides a comprehensive dataset of 45 years of U.S. federal contracts [5], emphasizing how crucial and voluminous this data is; however, they note that "users [of FPDS] primarily access this data through keyword searches" [5]. In Europe, the Tenders Electronic Daily (TED) portal similarly catalogs EU public contracts but relies on text search. In practice, many small businesses and researchers report difficulty discovering opportunities, since "95% of federal contracts are never posted on SAM.gov" and must be found through intelligence tools [3, 5]. Open contracting advocates (e.g., Open Contracting Partnership) stress that easier data access boosts competition. Our hybrid engine aligns with these goals by improving discoverability via smarter search.

Academic work on procurement analysis tends to focus on policy and economics (e.g., how open data affects competition [6, 5]), but technical studies on RFP search are scarce. One recent project (Rayo et al., 2025) developed a hybrid IR system for regulatory texts, combining BM25 with a fine-tuned Sentence Transformer [7]. They reported that the hybrid approach significantly outperformed standalone lexical or semantic methods (e.g., higher Recall@10 and MAP@10) [7], validating the value of a combined strategy. Our work follows this paradigm in the procurement domain.

3.2 Hybrid Information Retrieval Systems

Hybrid IR—fusing lexical and semantic signals—has garnered interest. Elastic's recent technical articles discuss "hybrid search" in Elasticsearch, noting that carefully merging keyword and vector results often "yields far better results than either lexical or semantic search would do on their own" [2]. Typical hybrid methods combine a keyword query (scored by BM25/TF-IDF) with a semantic (vector) query, then fuse results via techniques like Convex Combination (CC) or Reciprocal Rank Fusion (RRF) [2]. In CC, normalized scores from each branch are weighted and summed; in RRF, documents are scored by summing $1/(k + \text{rank})$ over ranks from each list [2]. Elastic's engineers note that RRF has the advantage of not requiring precise weight tuning [2]. These ideas inform our scoring design; we implemented Convex Combination with equal weights ($\alpha = 0.5$) as a baseline fusion strategy.

Production systems like Elasticsearch now support hybrid search natively, combining BM25 with dense vector queries and offering RRF fusion. We opted for a custom implementation to maintain flexibility over embedding model selection and fusion parameters, and to avoid enterprise licensing requirements.

4 System Design and Implementation

4.1 Architecture Overview

The search engine comprises four modules: Data Ingestion, Index Construction, Search, and Web Interface. The system ingests RFP data from SAM.gov's Opportunities API, storing results in a flat CSV file. At startup, the application builds two parallel in-memory indexes: a TF-IDF sparse matrix for lexical matching and a dense embedding matrix for semantic similarity. User queries are processed through both indexes, and results are fused using convex combination before being presented through a web interface [2].

4.2 Data Ingestion

The ingestion module (`ingest_sam.py`) fetches RFP data from the SAM.gov Opportunities API. For each opportunity, we extract: title and description text; issuing organization and parent agency hierarchy; NAICS and PSC classification codes; response deadline and place of performance; and links to full solicitation documents.

The API returns structured JSON, but description text often requires additional fetching from linked URLs. The ingestion script handles both inline descriptions and external description endpoints, stripping HTML tags and normalizing whitespace. Results are stored in a CSV file (`rfp.csv`) with one row per opportunity and columns for each extracted field. While a production system might use a relational vector database, the flat-file approach simplifies deployment and debugging for this prototype.

4.3 Index Construction

At application startup, the system builds two parallel indexes from the CSV data:

TF-IDF Index: We use scikit-learn's `TfidfVectorizer` configured with English stopword removal, a maximum vocabulary of 50,000 terms, and unigram/bigram features (n-gram range 1–2). For each document, we concatenate the title, organization name, parent agency path, NAICS codes, PSC codes, and description into a single text field. This combined text is transformed into a sparse TF-IDF vector. Query-time scoring uses cosine similarity computed via `linear_kernel`.

Semantic Index: We encode each document's combined text using the all-MiniLM-L6-v2 model from SentenceTransformers. This produces a dense numpy matrix of 384-dimensional embeddings, normalized to unit length. Because embeddings are pre-normalized, cosine similarity reduces to a simple dot product, enabling efficient computation via matrix multiplication.

Both indexes are held entirely in memory, enabling sub-second query response times. For our test corpus of approximately 50–1000 RFPs, index construction completes in under 30 seconds.

4.4 Search and Scoring

When a user submits a query, the system can operate in three modes: keyword-only (TF-IDF), semantic-only, or hybrid.

Keyword Search: The query is transformed using the same TF-IDF vectorizer, and cosine similarity is computed against all document vectors. Documents with zero similarity (no term overlap) are excluded.

Semantic Search: The query is encoded using the same SentenceTransformer model. Cosine similarity is computed via dot product against all document embeddings. This captures conceptual similarity even when vocabulary differs.

Hybrid Search: Both searches execute in parallel. Scores are min-max normalized within each result set to a $[0, 1]$ scale, ensuring both components contribute equally regardless of their raw score distributions. The final score uses convex combination:

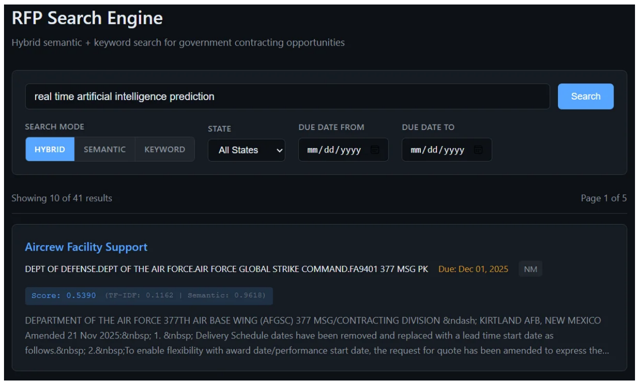
$$\text{HybridScore}(D) = \alpha \cdot \text{TF-IDF}_{\text{norm}}(D) + (1-\alpha) \cdot \text{Semantic}_{\text{norm}}(D) \tag{4}$$

We use $\alpha = 0.5$, giving equal weight to lexical and semantic signals. Documents appearing in only one result set receive a score of zero for the missing component, which penalizes documents that lack cross-method agreement.

4.5 User Interface

The web interface is built with FastAPI and Jinja2 templates. Key features include: search mode toggle allowing users to select Hybrid (default), Semantic, or Keyword mode; filters including dropdown for state/territory and date pickers for response deadline range; score transparency displaying both component scores (TF-IDF and semantic) alongside the combined score in hybrid mode; and pagination with 10 items per page.

The interface uses a dark theme designed for extended use, with clear visual hierarchy distinguishing metadata (agency, deadline, state) from description snippets. Figure 1 shows the search interface in action.



RFP Search Engine

Hybrid semantic + keyword search for government contracting opportunities

real time artificial intelligence prediction

Showing 10 of 41 results Page 1 of 5

Aircrew Facility Support

DEPT OF DEFENSE.DEPT OF THE AIR FORCE.AIR FORCE GLOBAL STRIKE COMMAND.FA9401 377 MSG PK Due: Dec 01, 2025 NM

Score: 0.5390 (TF-IDF: 0.1162 | Semantic: 0.9618)

DEPARTMENT OF THE AIR FORCE 377TH AIR BASE WING (AFGSC) 377 MSG/CONTRACTING DIVISION &ndash; KIRTLAND AFB, NEW MEXICO Amended 21 Nov 2025:&nbsp; 1. &nbsp; Delivery Schedule dates have been removed and replaced with a lead time start date as follows.&nbsp; 2.&nbsp;To enable flexibility with award date/performance start date, the request for quote has been amended to express the...

Figure 1: Search interface showing hybrid mode with score transparency. The result displays high semantic similarity (0.96) but low keyword match (0.12), illustrating how hybrid search surfaces relevant opportunities despite vocabulary mismatch.

4.6 Design Decisions and Trade-offs

TF-IDF vs. BM25: We use TF-IDF rather than BM25 for the lexical component. While BM25 offers improved handling of document length variation, TF-IDF integrates directly with scikit-learn's vectorization pipeline and is sufficient for demonstrating hybrid retrieval. The semantic component compensates for limitations in lexical matching regardless of which term-weighting scheme is used. BM25 remains a candidate for future optimization.

In-Memory vs. Persistent Index: Both indexes are rebuilt from CSV at each application startup rather than persisted to disk. This simplifies the implementation and ensures consistency, though it would not scale to millions of documents. For corpora beyond tens of thousands of RFPs, a production system would use persistent vector stores (e.g., FAISS, Pinecone) and inverted indexes (e.g., Elasticsearch, Whoosh).

Equal Fusion Weights: We chose $\alpha = 0.5$ rather than tuning weights empirically. Without labeled relevance data, any tuning would risk overfitting to subjective judgments. Equal weighting provides a neutral baseline that users can interpret: a high hybrid score means strong agreement from both methods.

Normalization Strategy: Min-max normalization ensures both score components fall in $[0, 1]$, but it means scores are relative within each query's result set. A document scoring 1.0 on TF-IDF is simply the best lexical match for that query, not an absolute measure of relevance.

5 Evaluation

5.1 Evaluation Approach

Standard IR evaluation measures include Precision@K, Recall@K, MAP (Mean Average Precision), MRR (Mean Reciprocal Rank), and nDCG (Normalized Discounted Cumulative Gain). Rayo et al. used Recall@10 and MAP@10 to demonstrate hybrid search gains over lexical or semantic methods alone [7]. However, these metrics require ground-truth relevance judgments—labeled query-document pairs indicating which results are correct.

Building a gold-standard for RFP search is challenging: relevance is context-dependent (an RFP relevant to one contractor may be irrelevant to another), and no public benchmark exists for procurement search. Without labeled data, we cannot compute standard IR metrics.

Instead, we employed three complementary evaluation methods: stakeholder interviews to understand real-world needs, comparative analysis across search modes, and quantitative measurement of retrieval differences between methods.

5.2 Stakeholder Interviews

We conducted informal interviews with stakeholders in the government technology space to understand current procurement discovery practices and pain points.

Sydney Lienemann, Deputy Cabinet Secretary for Administration at the New Mexico Environment Department (NMED), described the challenges government agencies face when searching for vendors and contractors. She noted that she is "hesitant to rely on terminology-based searches because terminology is inconsistent across agencies and systems"—a query that works for one portal may yield no results on another. This observation aligns directly with the vocabulary mismatch problem that motivates hybrid search.

However, Lienemann emphasized that terminology concerns are secondary to a more fundamental issue: "the fear of missing relevant RFPs outweighs terminology concerns." Her current workflow involves manually monitoring approximately 50 different procurement websites to ensure comprehensive coverage. This approach, while thorough, is time-intensive and unsustainable for smaller organizations without dedicated procurement staff.

A local govtech startup working on environmental data solutions shared their experience tracking relevant opportunities. The startup is actively monitoring RFPs including: California WaterTAP (RFP 25-020-280), a Water Technical Access Portal solicitation from the California State Water Resources Control Board; and Arizona ADEQ Environmental Hydrogeology Application (BPM007299), an environmental hydrogeology data analysis and visualization solicitation.

These opportunities were discovered through different channels—the California RFP through the state's eProcure system, the Arizona opportunity through an agency LinkedIn post before formal solicitation. Neither would appear in a SAM.gov search.

5.3 Comparative Analysis

We tested the system with queries designed to expose differences between search modes. All tests used a corpus of 55 RFPs ingested from SAM.gov, examining the top 10 results from each method.

Query: "environmental consulting"

Metric	Value
Keyword matches (TF-IDF > 0)	2 of 55 (3.6%)
Top-10 overlap (Keyword $\cap$ Semantic)	20%
Semantic rescue rate	80%

Only two documents contained the exact terms. The hybrid top-10 included 8 documents that keyword search would have missed entirely—demonstrating semantic search's value for vocabulary mismatch. Semantic search surfaced opportunities like "Region 3 Fuels Management" that relate to environmental work despite lacking the exact terminology.

Query: "water management"

Metric	Value
Keyword matches (TF-IDF > 0)	3 of 55 (5.5%)
Top-10 overlap (Keyword $\cap$ Semantic)	30%
Semantic rescue rate	80%

Keyword search correctly prioritized "Water System Upgrades Santa Fe National Forest." Semantic search additionally surfaced "Takini School Dry Pipe Sprinkler" and infrastructure contracts—conceptually related to water systems despite different terminology. Hybrid mode ranked the exact match first while including semantic expansions.

Query: "construction"

This query showed the starkest difference: only one document contained "construction," but semantic search identified related opportunities including "Quarters Renovation" and "High Speed Test Track" projects. Hybrid search rescued 9 of 10 results that keyword search alone would have missed.

Metric	Value
Keyword matches (TF-IDF > 0)	1 of 55 (1.8%)
Top-10 overlap (Keyword $\cap$ Semantic)	20%
Semantic rescue rate	90%

Query: "IT services"

Metric	Value
Keyword matches (TF-IDF > 0)	22 of 55 (40.0%)
Top-10 overlap (Keyword $\cap$ Semantic)	30%
Semantic rescue rate	0%

This illustrates the opposite case: "services" is a common term, so TF-IDF matched many documents. Semantic search correctly prioritized "Information Technology (IT) Specialized Personnel"—the most relevant result—which hybrid mode elevated to position 2. No semantic rescue was needed because keyword coverage was already high.

5.4 Quantitative Metrics

We computed aggregate metrics across 7 test queries to measure the practical difference between search modes.

Keyword Sparsity

Metric	Value
Average documents with TF-IDF > 0	15.8% of corpus
Range	1.8% to 40.0%

On average, keyword search found non-zero matches for fewer than 1 in 6 documents. This sparsity—caused by vocabulary mismatch—motivates semantic augmentation.

Top-10 Result Set Overlap

Comparison	Average Overlap
Keyword vs. Semantic	29%
Keyword vs. Hybrid	46%
Semantic vs. Hybrid	83%

The low overlap between keyword and semantic results (29%) confirms that the two methods retrieve substantially different document sets. Hybrid mode draws more heavily from semantic results (83% overlap) while still incorporating keyword matches (46% overlap).

Semantic Contribution: On average, 51% of hybrid top-10 results had zero TF-IDF scores—meaning keyword search alone would have missed them entirely. This "semantic rescue rate" quantifies the value added by embedding-based retrieval: half of all hybrid results are surfaced only because semantic search identified conceptual similarity that lexical matching missed.

5.5 Discussion

5.5.1 The Unified Platform Problem

Our stakeholder interviews revealed that vocabulary mismatch, while important, is only part of the RFP discovery challenge. As we hypothesized, the more fundamental problem is **fragmentation**: opportunities are scattered across federal, state, and municipal portals with no unified search interface.

Sydney Lienemann's workflow—manually checking 50 websites—represents a common but unsustainable approach. The govtech startup's experience tracking the California and Arizona opportunities illustrates the same issue: relevant RFPs may appear on state portals, agency websites, or even social media before formal solicitation.

Our prototype addresses the vocabulary mismatch problem through hybrid search but does not solve fragmentation. The system currently ingests only from SAM.gov (federal opportunities). A production system would need to aggregate data from: SAM.gov (federal); state procurement portals (50 states, varying APIs and formats); municipal procurement systems; agency-specific announcement channels; and private databases and providers.

The hybrid search architecture we demonstrate becomes more valuable as data sources multiply, because terminology varies not only across agencies but across levels of government. A unified procurement intelligence platform would combine multi-source aggregation with hybrid retrieval to address both fragmentation and vocabulary mismatch.

5.5.2 Implications for System Design

The stakeholder feedback suggests several priorities for future development: (1) Coverage over precision—users fear missing opportunities more than seeing irrelevant results, suggesting erring toward higher recall (lower α values favoring semantic expansion); (2) Transparency builds trust—displaying component scores helps users understand why results ranked as they did and refine their search strategy; (3) Filters are essential—state, deadline, and NAICS filters allow users to narrow results efficiently after broad semantic retrieval.

5.6 Limitations

Single data source: Our system only ingests federal opportunities from SAM.gov. The state-level RFPs discussed (California, Arizona) would require additional ingestion pipelines.

No ground-truth labels: Without expert relevance judgments, we cannot compute standard IR metrics (MAP, nDCG) that would allow comparison with benchmark systems.

Small user sample: Two stakeholder conversations cannot represent the diversity of procurement professionals.

Corpus size: Testing used 55 RFPs, primarily from New Mexico. System behavior and result quality may differ significantly with larger, more diverse corpora.

Temporal validity: RFP language evolves; our embedding model may not capture recent terminology shifts without fine-tuning.

6 Ethical Considerations

A search engine for public procurement must consider fairness and ethics. One issue is **bias in retrieval**. If the underlying data reflects historical inequalities, the search algorithm might inadvertently favor certain firms. For example, studies show that, despite increases in contracting dollars, the share of contracts won by women- and minority-owned businesses remains flat or declining [3]. If our system disproportionately ranks RFPs relevant to established incumbents, it could exacerbate this bias. To mitigate this, we ensured diverse test queries during development and implemented filters (e.g., set-aside categories, NAICS codes) that help disadvantaged groups find matching RFPs. We avoid inferring sensitive attributes of companies, but we recognize that embedding spaces can encode unintended correlations; this remains an area for ongoing monitoring in any production deployment.

Algorithmic transparency is a design priority. Our interface displays both TF-IDF and semantic scores for each result in hybrid mode, allowing users to understand why a document ranked highly. This transparency helps users calibrate trust—if a result has high semantic similarity but zero keyword match, users can judge whether the conceptual connection is meaningful or spurious. Opaque ranking systems risk reinforcing biases invisibly; score transparency provides at least partial accountability [7].

Environmental impact is another concern. Large language models consume significant energy. We consciously chose a smaller, efficient embedding model (MiniLM, 384 dimensions) to keep the carbon footprint low [11]. We also batch-process embeddings during ingestion and reuse them across queries to minimize redundant computation. These design choices—model size, precomputed embeddings, infrequent re-indexing—reduce energy consumption compared to query-time inference with larger models.

Finally, **data privacy and openness** matter. Our system only uses public RFP data (government contracts are public records). We ensured no inadvertent personally identifiable information (PII) is indexed—RFP documents rarely contain personal data, but this was verified during data inspection. We follow open data licenses: all external data sources are cited, and the project code is available for inspection. The project aligns with open science principles and aims to increase transparency in public procurement.

Legal considerations informed our technical choices. Web scraping federal government websites raises legal questions under the Computer Fraud and Abuse Act and site-specific terms of service. Rather than scrape procurement portals directly, we chose to use SAM.gov's official public API, which provides structured access to federal opportunities under clear terms of use. This approach ensures legal compliance while demonstrating that the system could scale to additional data sources that offer API access.

7 Conclusion

Hybrid search solves the vocabulary problem. It doesn't solve the fragmentation problem. Our prototype shows that combining lexical and semantic retrieval surfaces opportunities that either method alone would miss—but those opportunities still come from a single source.

The path forward is infrastructure: automating ingestion across federal, state, and municipal systems; scheduling pipelines that surface opportunities before deadlines close; and fine-tuning search based on what users actually bid on. The goal is a system where finding the right RFP is as simple as searching for it.

References

[1] Elastic. *Elasticsearch Scoring and the Explain API*. 2024. URL: https://www.elastic.co/search-labs/blog/elasticsearch-scoring-and-explain-api.

[2] Elastic. *Hybrid Search in Elasticsearch*. 2024. URL: https://www.elastic.co/search-labs/blog/hybrid-search-elasticsearch.

[3] HigherGov. *Small Business Trends in Federal Contracting*. 2022. URL: https://www.highergov.com/reports/small-business-trends-2022/.

[4] Manning, Christopher D., Raghavan, Prabhakar, and Schütze, Hinrich. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[5] Omari, Sami et al. "A comprehensive dataset of 45 years of US federal government contracts". In: *Scientific Data* (2025). URL: https://www.nature.com/articles/s41597-025-05714-1.

[6] Rauter, Thomas. *The Effect of Mandatory Disclosure on Market Efficiency: Evidence from Government Procurement*. Utah Winter Accounting Conference. 2020. URL: https://www.utah-wac.org/2020/Papers/rauter_UWAC.pdf.

[7] Rayo, Alejandro et al. *Hybrid Information Retrieval for Regulatory Texts*. 2025. arXiv: 2502.16767. URL: https://arxiv.org/abs/2502.16767.

[8] Reimers, Nils and Gurevych, Iryna. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*. 2019. URL: https://aclanthology.org/D19-1410/.

[9] Robertson, Stephen and Zaragoza, Hugo. "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Foundations and Trends in Information Retrieval* 3.4 (2009), pp. 333–389.

[10] Sentence-Transformers. *Semantic Search*. 2024. URL: https://sbert.net/examples/sentence_transformer/applications/semantic-search/README.html.

[11] Sentence-Transformers. *Sentence-Transformers Documentation*. 2024. URL: https://sbert.net/.

[12] World Bank. *Global Public Procurement Database*. 2023. URL: https://www.worldbank.org/en/topic/governance/brief/global-public-procurement-database.

A Replication Instructions

This appendix provides instructions for running the RFP search engine locally.

A.1 Prerequisites

- Python 3.10 or higher
- pip (Python package manager)
- A SAM.gov API key (free registration at `https://sam.gov/content/entity-registration`)

A.2 Installation

1. Clone the repository:

   ```
   git clone https://github.com/[your-username]/rfp-semantic-search.git
   cd rfp-semantic-search
   ```

2. Install dependencies:

   ```
   pip install fastapi uvicorn jinja2 pandas numpy scikit-learn sentence-transformers python-dotenv
       requests
   ```

3. Create a `.env` file with your SAM.gov API key:

   ```
   SAM_API_KEY=your_api_key_here
   ```

4. Ingest RFP data:

   ```
   python ingest_sam.py --posted-from 2024-01-01 --posted-to 2024-12-31 --max-records 200
   ```

5. Start the server:

   ```
   python -m uvicorn main:app --reload
   ```

6. Open `http://127.0.0.1:8000` in your browser.

A.3 Configuration Options

The ingestion script accepts several parameters:

Parameter	Description	Example
`--posted-from`	Start date for opportunities (YYYY-MM-DD)	2024-01-01
`--posted-to`	End date for opportunities (YYYY-MM-DD)	2024-12-31
`--max-records`	Maximum number of records to fetch	200
`--naics`	Filter by NAICS code	541620
`--state`	Filter by state code	CA

A.4 File Structure

After setup, your directory should contain:

```
rfp-semantic-search/
  main.py              # FastAPI application and search logic
  ingest_sam.py        # SAM.gov data ingestion script
  rfp.csv              # Ingested RFP data (generated)
  .env                 # API key configuration (create this)
  templates/
    search.html        # Web interface template
```

A.5 Troubleshooting

"uvicorn not recognized": Use `python -m uvicorn main:app --reload` instead of `uvicorn main:app --reload`.
"rfp.csv not found": Run the ingestion script first with valid date parameters.
Slow startup: The first run downloads the SentenceTransformer model (∼90MB). Subsequent runs use the cached model from `~/.cache/huggingface/`.
No results for queries: Ensure your ingested data actually contains relevant content. Try broad queries like "contract" or "services" to verify the system is working.

A.6 Verifying Installation

After starting the server, you should see:

```
Initialized indexes on [N] RFPs
INFO: Uvicorn running on http://127.0.0.1:8000
```

Navigate to `http://127.0.0.1:8000` and try a search query. If you see results with scores, the installation is successful.

B Code Architecture Overview

This appendix describes the organization of the codebase for developers who wish to extend or modify the system.

B.1 Module Overview

The system consists of two Python scripts and one HTML template:

File	Purpose
`main.py`	Search engine and web server
`ingest_sam.py`	Data ingestion from SAM.gov
`templates/search.html`	Web interface

B.2 main.py Architecture

B.2.1 Global State

The application maintains global state for performance (avoiding re-computation on each request):

```
df: pd.DataFrame           # RFP data with combined_text column
tfidf_vectorizer: TfidfVectorizer  # Fitted vectorizer
tfidf_matrix: sparse matrix        # Document TF-IDF vectors
embed_model: SentenceTransformer   # Loaded embedding model
embeddings: np.ndarray             # Document embedding matrix
```

B.2.2 Key Functions

B.2.3 Data Flow

1. User submits query via web form
2. `search_page()` receives HTTP request
3. `run_search()` dispatches based on mode parameter
4. For hybrid: `run_hybrid()` calls both `run_tfidf()` and `run_semantic()`
5. Scores normalized and combined: $\alpha \cdot \text{tfidf} + (1 - \alpha) \cdot \text{semantic}$

Function	Purpose
`init_indexes()`	Load CSV, build TF-IDF and embedding matrices
`normalize_scores(scores)`	Min-max normalize scores to $[0, 1]$ range
`run_tfidf(query, top_k)`	Execute TF-IDF search
`run_semantic(query, top_k)`	Execute embedding search
`run_hybrid(query, alpha, top_k)`	Combine both methods
`run_search(query, mode, top_k)`	Unified interface
`apply_filters(df, state, ...)`	Post-search filtering
`format_results(df)`	Convert to display dictionaries
`search_page()`	FastAPI endpoint

6. `apply_filters()` removes results not matching state/date criteria
7. `format_results()` prepares data for template
8. Jinja2 renders `search.html` with results

B.3 ingest_sam.py Architecture

B.3.1 Key Functions

Function	Purpose
`fetch_to_csv()`	Main ingestion loop with pagination
`build_params()`	Construct API query parameters
`flatten_notice(notice)`	Extract fields from SAM.gov JSON
`fetch_description_text(url)`	Retrieve full descriptions

B.3.2 API Interaction

The script calls SAM.gov's `/opportunities/v2/search` endpoint with pagination:

```
while offset < total_records and len(rows) < max_records:
    response = requests.get(BASE_URL, params=params)
    for notice in response['opportunitiesData']:
        rows.append(flatten_notice(notice))
    offset += limit
```

B.4 Extension Points

B.4.1 Adding a New Data Source

1. Create a new ingestion script following `ingest_sam.py` pattern
2. Ensure output CSV has the same column schema (especially id, title, description_text, state, response_date)
3. Merge with existing `rfp.csv` or replace entirely

B.4.2 Changing the Embedding Model

Modify the `SentenceTransformer()` initialization in `init_indexes()`:

```
# Current
model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")

# Alternative (larger, more accurate)
model = SentenceTransformer("sentence-transformers/all-mpnet-base-v2")
```

Note: Different models produce different embedding dimensions. Ensure the model produces normalized embeddings or add normalization.

B.4.3 Adjusting Fusion Weights

Change the alpha parameter in `run_hybrid()`:

```
# Current: equal weight
hybrid_scores[idx] = 0.5 * t_score + 0.5 * s_score

# Favor keyword matching
hybrid_scores[idx] = 0.7 * t_score + 0.3 * s_score

# Favor semantic matching
hybrid_scores[idx] = 0.3 * t_score + 0.7 * s_score
```

B.4.4 Adding New Filters

1. Add parameter to `search_page()` function signature
2. Pass parameter to template context
3. Extend `apply_filters()` with new filtering logic
4. Add UI element in `search.html`

Example for adding agency filter:

```
# In apply_filters()
if agency:
    df_res = df_res[df_res["organization_name"].str.contains(agency, case=False)]
```

B.5 Dependencies

Package	Version	Purpose
fastapi	≥0.100	Web framework
uvicorn	≥0.23	ASGI server
jinja2	≥3.0	HTML templating
pandas	≥2.0	Data manipulation
numpy	≥1.24	Numerical operations
scikit-learn	≥1.3	TF-IDF vectorization
sentence-transformers	≥2.2	Semantic embeddings
python-dotenv	≥1.0	Environment configuration
requests	≥2.31	HTTP client for API calls