Congressional Bill Analyzer - Technical Specification

Overview

A Python-based web application that fetches congressional bills from Congress.gov, processes them through AI analysis, and provides political ideology scoring from -10 (ultra-liberal) to +10 (ultra-conservative).

Architecture

Technology Stack

- **Backend**: Python 3.8+
- **Frontend**: HTML/JavaScript with Gradio interface
- **Vector Database**: ChromaDB
- **LLM**: OpenAI GPT-4/GPT-3.5-turbo
- **Data Source**: Congress.gov API v3

Required API Keys

1. **Congress.gov API Key** (Free)
   - Sign up at: https://api.data.gov/docs/congress/
   - Required for accessing congressional bill data
2. **OpenAI API Key**
   - Required for bill summarization and political scoring

Core Components

1. Bill Fetcher Module (`bill_fetcher.py`)

Purpose: Extract bill information from Congress.gov URLs and fetch full text via API

Key Functions:

```python
def parse_congress_url(url: str) -> dict:
    """Parse Congress.gov URL to extract bill identifiers"""
    # Extract: congress number, bill type, bill number
    # Example: https://www.congress.gov/bill/118th-congress/house-bill/1234
    # Returns: {"congress": "118", "bill_type": "house-bill", "number": "1234"}

def fetch_bill_metadata(congress: str, bill_type: str, number: str) -> dict:
    """Fetch bill metadata from Congress.gov API"""
    # API endpoint: https://api.congress.gov/v3/bill/{congress}/{bill_type}/{number}

def fetch_bill_text(congress: str, bill_type: str, number: str) -> str:
    """Fetch full bill text (introduced version only)"""
    # API endpoint: https://api.congress.gov/v3/bill/{congress}/{bill_type}/{number}/text
    # Filter for "Introduced" version
```

API Integration:

- Base URL: `https://api.congress.gov/v3/`
- Authentication: API key in header or query parameter
- Response format: JSON
- Rate limits: Standard government API limits (typically 1000/hour)

2. Text Processing Module (`text_processor.py`)

Purpose: Clean, chunk, and vectorize bill text for analysis

Key Functions:

python

```
def clean_bill_text(raw_text: str) -> str:
    """Remove formatting artifacts, preserve structure"""
    # Remove XML/HTML tags, excessive whitespace
    # Preserve section headers and numbering

def extract_sections(text: str) -> list:
    """Parse bill into logical sections"""
    # Identify: Title, Findings, Sections, Subsections
    # Return structured data with section headers and content

def chunk_text(text: str, chunk_size: int = 1000) -> list:
    """Split text into overlapping chunks for vectorization"""
    # Preserve sentence boundaries
    # Maintain context with overlapping chunks
```

Vector Store Integration:

python

```
def vectorize_bill(bill_text: str, bill_id: str) -> None:
    """Store bill chunks in ChromaDB with metadata"""
    # Embedding model: OpenAI text-embedding-ada-002
    # Metadata: bill_id, section, chunk_index, congress, bill_type
```

3. AI Analysis Module (`ai_analyzer.py`)

Purpose: Generate summaries and political ideology scores using LLM

Summarization Functions:

python

```
def generate_executive_summary(bill_text: str, metadata: dict) -> str:
    """Create high-level bill summary"""
    # Length: 200-300 words
    # Include: purpose, key provisions, affected parties

def generate_section_breakdown(sections: list) -> list:
    """Analyze each section individually"""
    # For each section: summary, key changes, implications
    # Return structured data with section-by-section analysis
```

Political Scoring System:

python

```
def score_political_ideology(bill_text: str, metadata: dict) -> dict:
    """Generate political ideology score (-10 to +10)"""
    # Return: {"score": float, "reasoning": str, "confidence": float}
```

4. Political Scoring Framework

Theoretical Foundation: Based on established political science research, including DW-NOMINATE methodology and GovTrack ideology analysis.

Scoring Dimensions:

1. **Economic Policy** (Weight: 30%)
   - Liberal indicators: Increased government spending, progressive taxation, economic regulation, social programs
   - Conservative indicators: Tax cuts, deregulation, free market solutions, reduced government spending
2. **Role of Government** (Weight: 25%)
   - Liberal indicators: Expanded federal programs, increased oversight, government intervention
   - Conservative indicators: States' rights, limited government, private sector solutions
3. **Social Policy** (Weight: 20%)
   - Liberal indicators: Civil rights expansions, social justice provisions, inclusivity measures
   - Conservative indicators: Traditional values, law and order, religious freedom protections
4. **Regulatory Approach** (Weight: 15%)
   - Liberal indicators: Environmental regulations, consumer protections, safety standards
   - Conservative indicators: Business-friendly policies, reduced compliance burden
5. **Fiscal Policy** (Weight: 10%)

- Liberal indicators: Deficit spending for social programs, progressive revenue
- Conservative indicators: Balanced budgets, debt reduction, fiscal responsibility

Scoring Methodology:

```python
IDEOLOGY_PROMPT = """
Analyze this congressional bill and score its political ideology on a scale of -10 to +10:
- -10: Ultra-liberal (maximum government intervention, progressive social policies)
- -5: Liberal (increased government role, social programs)
- 0: Moderate/Bipartisan
- +5: Conservative (limited government, traditional approaches)
- +10: Ultra-conservative (minimal government, strong traditional values)

Consider these factors:
1. Economic policy direction (spending, taxation, regulation)
2. Role of government (expansion vs. limitation)
3. Social policy implications
4. Regulatory approach
5. Fiscal impact

Bill text: {bill_text}

Provide:
1. Overall score (-10 to +10)
2. Detailed reasoning (200 words)
3. Confidence level (0-100%)
4. Key phrases that influenced the score
"""
```

5. Gradio Interface (`app.py`)

User Interface Components:

```python
def create_interface():
    with gr.Blocks(title="Congressional Bill Analyzer") as app:
        gr.Markdown("# Congressional Bill Analyzer")

        with gr.Row():
            url_input = gr.Textbox(
                label="Congress.gov Bill URL",
                placeholder="https://www.congress.gov/bill/118th-congress/house-bill/1234"
            )
            analyze_btn = gr.Button("Analyze Bill")

        with gr.Row():
            with gr.Column(scale=2):
                summary_output = gr.Markdown(label="Executive Summary")
                sections_output = gr.JSON(label="Section-by-Section Analysis")

            with gr.Column(scale=1):
                score_output = gr.Number(label="Political Ideology Score")
                score_reasoning = gr.Markdown(label="Scoring Rationale")
                confidence_output = gr.Number(label="Confidence Level")
```

Processing Pipeline:

```python
def analyze_bill(url: str) -> tuple:
    """Main processing pipeline"""
    try:
        # 1. Parse URL and validate
        bill_info = parse_congress_url(url)

        # 2. Fetch bill data
        metadata = fetch_bill_metadata(**bill_info)
        bill_text = fetch_bill_text(**bill_info)

        # 3. Process text
        clean_text = clean_bill_text(bill_text)
        sections = extract_sections(clean_text)

        # 4. Store in vector database
        vectorize_bill(clean_text, bill_info['bill_id'])

        # 5. Generate analysis
        summary = generate_executive_summary(clean_text, metadata)
        section_analysis = generate_section_breakdown(sections)
        ideology_score = score_political_ideology(clean_text, metadata)

        return summary, section_analysis, ideology_score

    except Exception as e:
        return f"Error: {str(e)}", {}, {}
```

Data Models

Bill Metadata

```python
@dataclass
class BillMetadata:
    congress: str
    bill_type: str  # "house-bill", "senate-bill", etc.
    number: str
    title: str
    sponsor: str
    introduced_date: str
    committees: list
    subjects: list
    url: str
```

Analysis Results

```python
@dataclass
class BillAnalysis:
    bill_id: str
    executive_summary: str
    section_breakdown: list
    ideology_score: float
    score_reasoning: str
    confidence: float
    analysis_date: str
```

File Structure

```
congressional_bill_analyzer/
├── src/
│   ├── __init__.py
│   ├── bill_fetcher.py
│   ├── text_processor.py
│   ├── ai_analyzer.py
│   └── app.py
├── data/
│   └── chromadb/  # Vector database storage
├── config/
│   ├── settings.py
│   └── prompts.py
├── tests/
│   ├── test_bill_fetcher.py
│   ├── test_text_processor.py
│   └── test_ai_analyzer.py
├── requirements.txt
├── .env.example
└── README.md
```

Installation & Setup

Dependencies (`requirements.txt`)

```
gradio>=4.0.0
chromadb>=0.4.0
openai>=1.0.0
requests>=2.28.0
python-dotenv>=1.0.0
beautifulsoup4>=4.11.0
sentence-transformers>=2.2.0
numpy>=1.21.0
pandas>=1.5.0
```

Environment Variables (`.env`)

```
CONGRESS_API_KEY=your_congress_gov_api_key
OPENAI_API_KEY=your_openai_api_key
CHROMA_DB_PATH=./data/chromadb
LOG_LEVEL=INFO
```

Quick Start

```bash
# Install dependencies
pip install -r requirements.txt

# Set up environment variables
cp .env.example .env
# Edit .env with your API keys

# Run the application
python src/app.py
```

Error Handling

Common Error Scenarios

1. **Invalid Congress.gov URL**: Validate URL format and extract identifiers
2. **Bill not found**: Handle 404 responses from API gracefully
3. **API rate limits**: Implement retry logic with exponential backoff
4. **Text processing failures**: Fallback to simpler analysis methods
5. **OpenAI API errors**: Handle rate limits and token limits

User Feedback

- Progress indicators during processing
- Clear error messages with suggested fixes
- Validation of input URLs before processing

Performance Considerations

Optimization Strategies

1. **Caching**: Store processed bills in ChromaDB to avoid reprocessing
2. **Chunking**: Process large bills in sections to stay within token limits
3. **Async Processing**: Use background processing for time-intensive operations
4. **Response Streaming**: Stream analysis results as they become available

Scalability Notes

- Current design targets single-user deployment
- For multi-user: add session management and request queuing
- Consider database migrations for production deployment

Future Enhancements

Version 2.0 Features

1. **Multi-dimensional Scoring**: Break down ideology score by policy area
2. **Historical Comparison**: Compare current bill to similar past legislation
3. **Amendment Tracking**: Analyze changes through legislative process
4. **Batch Processing**: Analyze multiple bills simultaneously
5. **Export Functionality**: PDF reports and data export options

Advanced Analysis

1. **Sentiment Analysis**: Tone and rhetoric analysis
2. **Stakeholder Impact**: Identify affected groups and interests
3. **Cost Estimation**: Integrate CBO scoring when available
4. **Prediction Models**: Likelihood of passage based on historical data