**Overall Architecture**

* The application follows a client-server architecture with distinct frontend (Streamlit) and backend components
* Uses AWS Bedrock to access Claude 3 LLM for grammatical corrections and text standardization
* Combines rule-based regex pattern matching with AI-powered text processing
* Implements configurable style formats that can be saved and reused

**Frontend (Streamlit)**

* **Main UI Components**:
  + Style Format Manager (sidebar)
  + Rules Selection interface (categorized by type)
  + Target Format Configuration (for selected rules)
  + Document Upload and Processing section
  + Results display with download functionality
* **Key Frontend Features**:
  + Custom styling with CSS for improved UX
  + Tab-based organization of rules by category (Time Comparisons, Time Periods, Formatting, Ranges)
  + Persistent storage of style formats in JSON file
  + Visual progress tracking during document processing
  + Interactive rule selection with tooltips showing examples

**Backend**

* **Core Components**:
  + AWS Bedrock client configuration (using boto3)
  + LLM integration with Claude 3 Sonnet model
  + Extensive regex pattern library for text standardization
  + Document processing pipeline with Microsoft Word manipulation
* **Key Backend Functions**:
  + invoke\_claude\_3\_with\_text: Interfaces with AWS Bedrock to access Claude 3
  + process\_docx\_with\_llm: Main processing pipeline for documents
  + apply\_regex\_replacements: Handles pattern-based text standardization
  + process\_paragraph: Processes individual document paragraphs
  + compare\_documents: Creates track changes between original and corrected documents

**Data Flow**

1. User uploads a DOCX document and selects style rules
2. Frontend sends document and selected rules to backend
3. Backend creates temporary files and extracts document content
4. Document is processed paragraph by paragraph:
   * For each paragraph, LLM-based grammar correction is applied first
   * Then regex-based standardization is applied for patterns
   * Number formatting and word conversion are applied last
5. Modified document is saved with track changes
6. Processed document is sent back to frontend for download

**Configuration Management**

* Uses a separate config.py module to handle application configuration
* Configuration stored in config.ini file
* Provides getters for AWS and LLM configuration parameters
* Creates default configuration if none exists

**Rule-Based Processing System**

* Rules are defined in REPLACEMENT\_RULES dictionary
* Each rule contains:
  + Examples of patterns to match
  + Target formatting options
  + Default target format
  + Optional dynamic regex pattern for complex matches
* Rules are categorized by type (Time Comparisons, Time Periods, Formatting, Ranges)
* User can select which rules to apply and customize target formats

**Document Comparison and Tracking**

* Uses Spire.Doc library for document comparison
* Creates a track changes document showing differences
* Removes evaluation warnings from final output
* Provides statistics on number of changes made

**Error Handling**

* Try-except blocks for LLM API calls and document processing
* Graceful fallback when LLM processing fails
* Proper resource cleanup with temporary files
* User-friendly error messages in the UI