Automated Data Collection and Preparation Pipeline

Parikshak.ai Data Engineering Intern Challenge
This project automates the end-to-end process of collecting, cleaning, and annotating data for machine learning purposes. By providing a specific industry or domain keyword, the pipeline automatically gathers relevant URLs, extracts and cleans the text content, structures the data into a clean **CSV schema**, and produces annotated examples ready for AI/ML workflows. This approach emphasizes automation to ensure scalability, reproducibility, and efficiency, mirroring real-world data engineering practices.

Project Overview

This project implements a fully automated pipeline for:

- **Data Collection** – Web scraping and web search to gather raw text samples.
- **Data Cleaning** – Removing noise, duplicates, HTML artifacts, and normalizing text.
- **Data Annotation** – Adding structured labels for downstream machine learning training.

Instead of manually downloading and cleaning datasets, this pipeline is designed to be end-to-end automated. By providing an industry/domain keyword (e.g., "Engineering Services"), the system automatically:

- Finds relevant URLs using refined queries.
- Extracts and cleans text content.
- Structures data into a clean **CSV schema**.
- Produces annotated examples ready for AI/ML workflows.

This design reflects real-world data engineering practice where automation ensures scalability, reproducibility, and efficiency.

Why Automation?

Traditional Approach

- Manual scraping, cleaning, and annotation.
- Time-consuming and error-prone.
- Hard to scale when datasets change or grow.

Automated Approach

- One function call produces raw, cleaned, and annotated **CSV datasets**.
- Works for any industry/domain with minimal changes.
- Consistent preprocessing and annotation logic.
- Schema designed for downstream ML pipelines.
- Automation ensures that adding a new dataset requires only changing the industry keyword.

Tools and Libraries Used

- **Python** (core language)
- **Requests / BeautifulSoup** (web scraping)
- **SerpAPI** (web searching - free tier)
- **OpenAI GPT-4o-mini** (text cleaning and annotation)
- **Pandas** (data handling and storage)
- **python-dotenv** (environment management)

Performance Metrics

Test Case: "Engineering" Domain

- **Data Collected**: 87 unique URLs

- **Success Rate**: 70+ URLs successfully scraped (accounting for connection failures)
- **Total Runtime**: 15 minutes
    - Scraping: 3 minutes
    - Cleaning: 7 minutes
    - Annotation: 5 minutes
- **Total Cost**: $0.31 (GPT-4o-mini API usage)
- **Cost per Article**: ~$0.004

Pipeline Efficiency

- **Search Coverage**: 10 different query patterns per domain
- **Error Recovery**: 3 retry attempts with exponential backoff
- **Rate Limiting**: Built-in delays to respect server limits

Technical Challenges and Solutions

1. Connection Reliability Issues

Challenge: Many websites block automated requests or have unstable connections **Solution**:

- Implemented retry logic with exponential backoff (2, 4, 6 second delays)
- Proper User-Agent headers to appear as legitimate browser traffic
- Graceful error handling that continues pipeline execution

2. Content Quality Variation

Challenge: Scraped content varies wildly in format and quality **Solution**:

- LLM-powered cleaning that understands context rather than rigid regex rules
- Standardized output length (1000 characters) for consistent dataset structure
- Removal of navigation, boilerplate, and HTML artifacts through intelligent parsing

3. Scalable Annotation Schema

Challenge: Hard-coded annotation categories don't work across different domains **Solution**:

- Dynamic tag generation: first LLM call identifies relevant annotation categories
- Domain-aware annotation: tags adapt to content type (startups, market trends, technologies)
- Structured output format ready for ML training

4. API Cost Management

Challenge: LLM APIs can become expensive at scale **Solution**:

- Selected GPT-4o-mini for optimal cost/performance ratio
- Efficient prompting to minimize token usage
- Free tier SerpAPI for web search functionality

Data Pipeline Outputs

Raw Dataset (rawdata.csv)

- URL, title, raw scraped text
- 87 entries with full content

Cleaned Dataset (cleaned_data.csv)

- URL, title, cleaned text (1000 chars each)
- Normalized formatting, HTML removed

Annotated Dataset (annotated_data.csv)

- All cleaned data plus 4 annotation columns
- Ready for ML model training

Key Technical Decisions

1. LLM-Powered Cleaning vs Rule-Based Chose LLM approach because it handles edge cases and context better than regex patterns, despite slightly higher cost.

2. Dynamic vs Static Annotation Schema Implemented dynamic tag generation so the same pipeline works for any industry without manual reconfiguration.
3. Individual vs Batch API Calls Current implementation uses individual calls for simplicity and error isolation. Batch processing could reduce time roughly by 20-30%.

Scalability Considerations

Current Bottlenecks:

- Individual API calls
- Connection timeouts for certain domains

Optimization Opportunities:

- Batch annotation processing (3-5x speed improvement)
- Async/parallel processing for web scraping
- Local caching for previously processed URLs

Real-World Applications

This pipeline architecture directly applies to:

- **Recruitment Data**: Job postings, company profiles, industry reports
- **Market Research**: Competitor analysis, trend identification
- **Content Curation**: News articles, blog posts, technical documentation
- **Training Data Generation**: Domain-specific datasets for fine-tuning ML models

Usage

```
# Complete pipeline in one function call
df = create_df("Software Engineering")  # Change domain as needed
```

The automation ensures that switching to a new domain (e.g., "Healthcare", "Finance") requires only changing the input parameter while maintaining consistent data quality and structure.

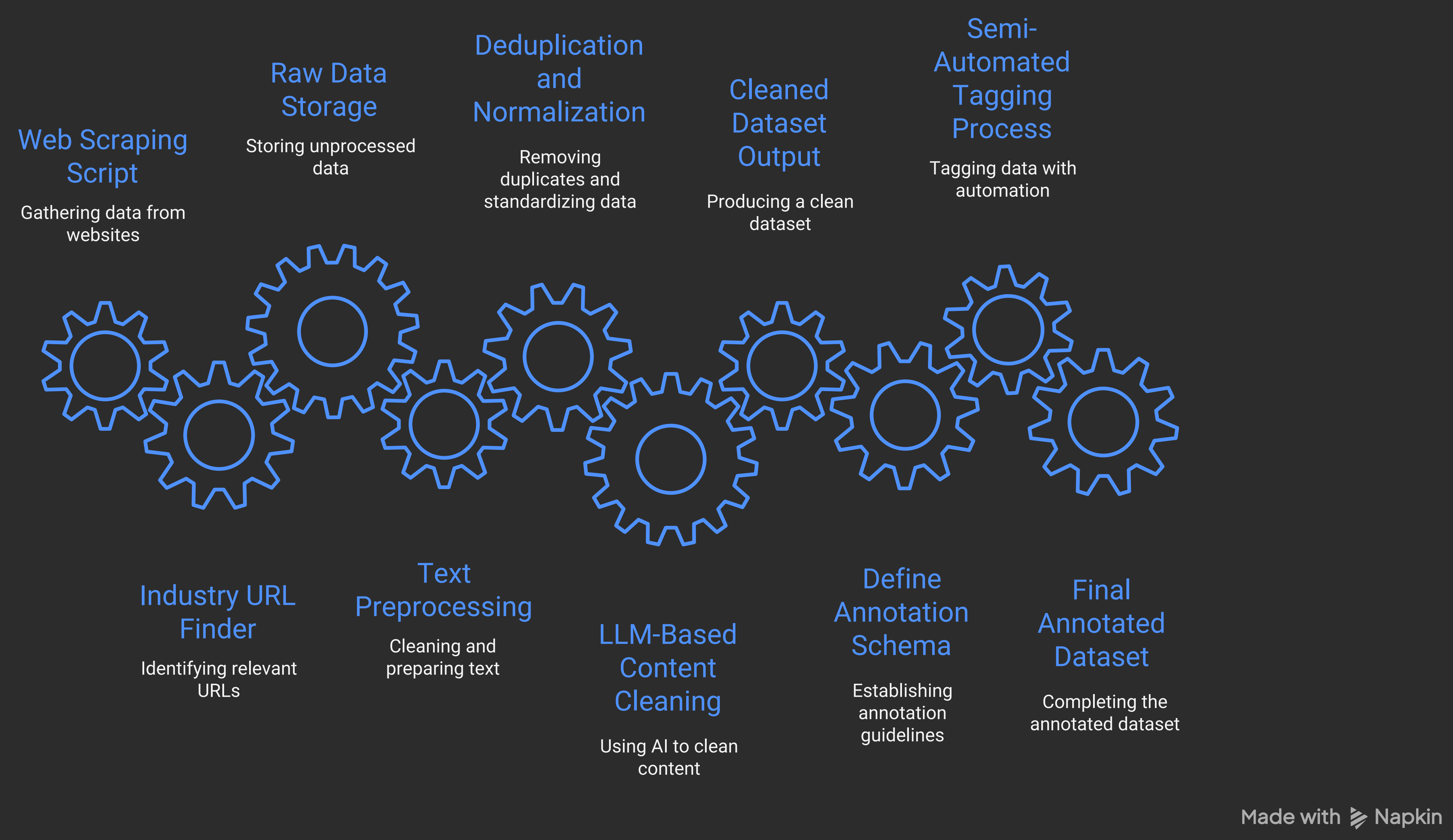
Repository Structure

```
├── Scrapper.ipynb     # URL discovery and content extraction
├── Cleaner.ipynb      # Text normalization and cleaning
├── Annotater.ipynb    # Dynamic annotation generation
├── rawdata.csv        # Original scraped content
├── cleaned_data.csv   # Processed and normalized text
└── README.md          # This documentation
```

This project demonstrates production-ready data engineering practices with emphasis on automation, error handling, and cost efficiency suitable for real-world deployment

Workflow Diagram

Data Engineering Pipeline Workflow



More About Me

If you’d like to explore more of my work or connect:

- LinkedIn
- GitHub