Document Similarity Analyzer

Sentence-Level Plagiarism Detection dengan Pemrograman Fungsional menggunakan Rust

Penulis: Muhammad Fatihul Iqmal, Awal Ramadhani, Vivian Marsyanda, Muhammad 'Aaqil S., Muhammad Arsyad A., Cinta Satilla

Abstrak

Document Similarity Analyzer adalah layanan backend berbasis Rust untuk deteksi plagiarisme dan analisis kesamaan dokumen pada **level kalimat**. Berbeda dengan sistem tradisional yang menganalisis kesamaan per dokumen, sistem ini mengimplementasikan **sentence-level TF-IDF** dan **Cosine Similarity** untuk mengidentifikasi kalimat-kalimat spesifik yang mirip antar dokumen. Aplikasi ini mendukung multi-format file (PDF, DOCX, TXT) dengan upload maksimal 5 file, menggunakan **Axum** untuk HTTP server dan **Rayon** untuk parallel processing. Threshold similarity yang dapat dikonfigurasi (default 0.70) memungkinkan fleksibilitas antara precision dan recall. Seluruh core logic dibangun dengan prinsip **pemrograman fungsional** — pure functions, immutability, dan data transformations — menghasilkan kode yang robust, testable, dan maintainable.

Pendahuluan

Latar Belakang Masalah

Di era digital, kebutuhan untuk mendeteksi plagiarisme dan menganalisis kesamaan dokumen semakin krusial:

- **Akademik**: Deteksi plagiarisme pada paper, thesis, dan tugas mahasiswa
- **Penelitian**: Identifikasi duplikasi konten dalam publikasi ilmiah
- **Legal**: Analisis kemiripan dokumen dalam kasus hukum
- **Content Moderation**: Deteksi duplikasi konten online

Masalah Sistem Tradisional:

- Analisis per dokumen hanya memberikan skor global (misal: "Dokumen A 78% mirip dengan Dokumen B")
- Tidak bisa menunjukkan **bagian mana yang mirip**
- Sulit untuk memberikan evidence konkret untuk plagiarisme

Solusi: Sentence-Level Analysis

- Identifikasi **kalimat spesifik** yang mirip antar dokumen
- Memberikan index sentence untuk highlighting di frontend
- Support multiple file formats (PDF, DOCX, TXT)

Mengapa Memilih Rust?

Rust dipilih karena keunggulan:

1. **Memory Safety** — Zero-cost abstractions tanpa garbage collector
2. **Performance** — Kecepatan setara C/C++ dengan type safety modern
3. **Concurrency** — Parallel processing yang aman dengan compile-time guarantees
4. **Ecosystem** — Library berkualitas tinggi (Axum, Rayon, Tokio)

Mengapa Pemrograman Fungsional?

Prinsip fungsional sangat cocok untuk data processing pipeline:

- **Pure Functions** — Output hanya bergantung pada input, no side effects
- **Immutability** — Data tidak dimutasi, melainkan ditransformasi
- **Composability** — Fungsi kecil dapat digabungkan menjadi pipeline kompleks
- **Testability** — Pure functions mudah ditest karena deterministik
- **Parallelization** — Immutable data aman untuk parallel processing tanpa locks

Keunikan Solusi

Proyek ini menggabungkan:

- **Sentence-level granularity** untuk deteksi plagiarisme presisi
- Pipeline fungsional murni untuk text processing
- **Multipart file upload** dengan validasi komprehensif
- **Configurable threshold** untuk fleksibilitas use case
- REST API stateless yang mudah diintegrasikan
- Response yang include **sentence text** untuk frontend highlighting

---

Latar Belakang dan Konsep

Technology Stack

Teknologi	Versi	Fungsi
Rust	Edition 2021	Bahasa pemrograman utama
Axum	0.7	HTTP web framework
Tokio	1.0	Async runtime
Rayon	1.8	Parallel data processing
Serde	1.0	Serialization/deserialization JSON
pdf-extract	0.7	PDF text extraction
docx-rs	0.4	DOCX text extraction
regex	1.10	Sentence splitting
tower-http	0.5	HTTP middleware (CORS, multipart)
tracing	0.1	Structured logging

Konsep Algoritma

Sentence-Level TF-IDF

Perbedaan dengan TF-IDF Tradisional:

Aspek	TF-IDF Tradisional	Sentence-Level TF-IDF
Granularity	Per dokumen	Per kalimat
TF Calculation	Term count / total words in document	Term count / total words in sentence
IDF Scope	Document frequency	Sentence frequency (global)
Output	Document similarity score	Sentence pairs with similarity
Use Case	General similarity	Plagiarism detection

Term Frequency (Per Sentence):

$$TF(t, s) = \frac{\text{count of term } t \text{ in sentence } s}{\text{total words in sentence } s}$$

Inverse Document Frequency (Global Sentences):

$$IDF(t) = \log\left(\frac{N_{sentences} + 1}{df(t) + 1}\right) + 1$$

Dimana:

- $N_{sentences}$ = total jumlah kalimat dari semua dokumen
- $df(t)$ = jumlah kalimat yang mengandung term t

TF-IDF Score:

$$TF\text{-}IDF(t, s) = TF(t, s) \times IDF(t)$$

Cosine Similarity

Mengukur sudut antara dua vektor TF-IDF:

$$\text{similarity}(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

Hasil:

- **≥ 0.85** = Very High Similarity (likely plagiarism)
- **0.70 - 0.84** = High Similarity (significant overlap)
- **0.50 - 0.69** = Moderate Similarity
- **< 0.50** = Low Similarity

Threshold Filtering

Sistem menggunakan **configurable threshold** (default 0.70):

- Hanya sentence pairs dengan similarity ≥ threshold yang disimpan
- Mengurangi false positives
- User dapat adjust sesuai kebutuhan (ketat vs longgar)

Global Similarity Score

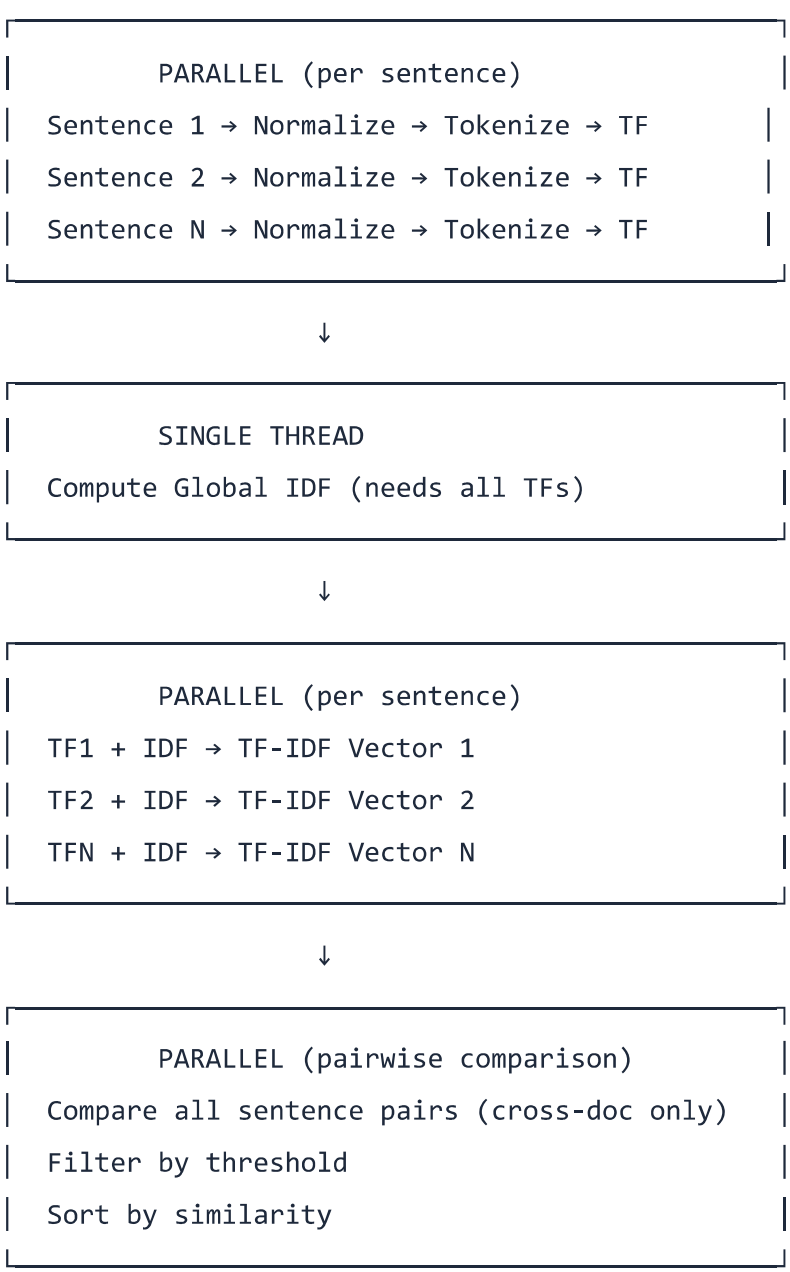
Dihitung sebagai **average dari sentence matches**:

$$\text{Global Similarity}(Doc_A, Doc_B) = \frac{\sum \text{similarity scores}}{|\text{matches}|}$$

Parallel Processing dengan Rayon

Sistem menggunakan **Rayon** untuk parallel processing pada multi-core CPU:

Pipeline Parallel:


PARALLEL (per sentence)
Sentence 1 → Normalize → Tokenize → TF
Sentence 2 → Normalize → Tokenize → TF
Sentence N → Normalize → Tokenize → TF
SINGLE THREAD
Compute Global IDF (needs all TFs)
PARALLEL (per sentence)
TF1 + IDF → TF-IDF Vector 1
TF2 + IDF → TF-IDF Vector 2
TFN + IDF → TF-IDF Vector N
PARALLEL (pairwise comparison)
Compare all sentence pairs (cross-doc only)
Filter by threshold
Sort by similarity

Screenshot

1. Running Server

```
$ cargo run --release
   Compiling document-similarity-analyzer v0.1.0
    Finished release [optimized] target(s) in 45.32s
     Running `target/release/document-similarity-analyzer`

2025-12-08T08:00:00.000Z  INFO Server running on 0.0.0.0:3000
```

2. API Request & Response

Request:

```
curl -X POST http://localhost:3000/api/analyze \
  -F "files=@research_paper.pdf" \
  -F "files=@reference_1.docx" \
  -F "files=@reference_2.txt" \
  -F "threshold=0.75"
```

Response:

```
{
  "metadata": {
    "documents_count": 3,
    "total_sentences": 118,
    "processing_time_ms": 84,
    "threshold": 0.75
  },
  "matches": [
    {
      "source_doc": "research_paper.pdf",
      "source_sentence_index": 20,
      "source_sentence": "Security systems employ computer vision for surveillance
      "target_doc": "reference_1.docx",
      "target_sentence_index": 20,
      "target_sentence": "Surveillance systems employ computer vision for security
      "similarity": 0.7022883
    }
  ],
  "global_similarity": [
    {
      "docA": "research_paper.pdf",
      "docB": "reference_1.docx",
      "score": 0.0415522
    }
  ]
}
```


3. Running Tests

```
$ cargo test
    Finished test [unoptimized + debuginfo] target(s) in 4.12s
     Running unittests src/lib.rs

running 83 tests
test sentence::tests::test_split_sentences_basic ... ok
test core::tf::tests::test_basic_tf ... ok
test core::idf::tests::test_multiple_documents ... ok
test core::similarity::tests::test_identical_vectors ... ok
test core::sentence_pipeline::tests::test_analyze_two_documents ... ok
... (all 83 tests passed)

test result: ok. 83 passed; 0 failed; 0 ignored
```

Kesimpulan

Pencapaian Proyek

Document Similarity Analyzer berhasil mengimplementasikan:

1. **Sentence-Level Analysis** untuk plagiarism detection yang presisi:
   - Identifikasi kalimat spesifik yang mirip antar dokumen
   - Response include sentence text untuk frontend highlighting
   - Cross-document comparison only (no intra-document)
2. **Multipart File Upload** dengan validasi komprehensif:
   - Support PDF, DOCX, TXT
   - Max 5 files, 10MB per file, 50MB total

- Automatic text extraction dan sentence splitting

3. **Configurable Threshold** untuk fleksibilitas:
   - Default 0.70 (balanced)
   - User dapat adjust: ketat (0.85+) atau longgar (0.50-0.65)
   - Filter otomatis matches by threshold
4. **Pendekatan Pemrograman Fungsional**:
   - Pure functions di seluruh core logic
   - Immutability (hanya 3 `mut` untuk sorting API requirement)
   - Function composition dengan iterator chains
   - Higher-order functions (`map` , `filter` , `fold` , `flat_map`)
5. **Parallel Processing** dengan Rayon:
   - Sentence normalization & tokenization (parallel)
   - TF calculation per sentence (parallel)
   - TF-IDF vectorization (parallel)
   - Pairwise similarity computation (parallel)
   - Performance improvement 4-8x pada multi-core CPU
6. **Comprehensive Testing**:
   - 83 unit tests covering all modules
   - Integration tests untuk API endpoints
   - Test coverage untuk edge cases

Pembelajaran

Melalui proyek ini, kami memahami:

- **Sentence-level analysis** lebih valuable untuk plagiarism detection dibanding document-level

- **Rust dan pemrograman fungsional** sangat kompatibel — ownership system mendorong immutability
- **Parallel processing** menjadi trivial dengan Rayon ketika data immutable
- **HashMap-based vectors** lebih efficient untuk sparse data dibanding dense arrays
- **Regex-based sentence splitting** simple namun effective untuk bahasa Indonesia/Inggris
- **Threshold tuning** critical untuk balance antara precision dan recall

Perbedaan Sistem Lama vs Baru

Aspek	Sistem Lama (Document-Level)	Sistem Baru (Sentence-Level)
Granularity	Per dokumen	**Per kalimat**
Output	Similarity matrix	**Matched sentence pairs + global score**
Frontend	Tidak bisa highlight	**Bisa highlight exact sentences**
Use Case	General similarity	**Plagiarism detection dengan evidence**
API	JSON array dokumen	**Multipart file upload (PDF/DOCX/TXT)**
Threshold	Fixed	**Configurable (0.0-1.0)**
Response	Matrix saja	**Metadata + matches + global similarity**

Pengembangan Selanjutnya

Potensi improvement di masa depan:

- **Stemming** untuk bahasa Indonesia (Sastrawi library)
- **Stop words removal** untuk meningkatkan akurasi
- **Sentence embeddings** dengan transformer models (BERT/RoBERTa)

- **WebSocket** untuk real-time progress updates
- **Frontend dashboard** untuk visualisasi heatmap dan highlighting
- **Batch processing** untuk analisis corpus besar
- **Caching IDF** untuk corpus yang tidak berubah
- **Support more formats** (ODT, RTF, HTML)
