Bank-Grade Translation Comparison Utility - Full Development Spec and Prompt Pack

Generated: 2025-08-21 03:23:46 Scope: Enterprise-safe, on-prem, auditable

1) Executive Summary

Build a compliant translation comparison utility for banking/legal PDFs. Inputs are a German source PDF and an English target PDF derived from Gemini's intermediate HTML. The tool extracts structure, aligns content block-by-block, computes quality scores (METEOR, BLEU, COMET-proxy), flags missing/extra sections, assigns risk severity, and generates auditor-ready HTML/PDF and JSON reports. Only enterprise-approved libraries are used; no external APIs.

2) Enterprise Constraints and Approved Libraries

PDF Handling (choose per stack; avoid mixing stacks unless needed):

- Python: pdfplumber, pdfminer.six, PyPDF2
- Java: Apache PDFBox (if JVM stack is preferred)

HTML/XML Parsing:

- beautifulsoup4, lxml (offline, no network)

NLP / Similarity (offline):

- NLTK (+ WordNet for METEOR), sacrebleu (BLEU)
- scikit-learn for TF-IDF + cosine similarity (or pure NumPy if sklearn restricted)
- No cloud APIs, no unvetted models; embeddings only if provided by in-house service

Templating and Reports:

- jinja2 (HTML), reportlab (PDF), or ops-approved wkhtmltopdf/WeasyPrint/PrinceXML binaries

General:

- logging, json, yaml (config), dataclasses or pydantic (optional)
- Strict pinning of versions; reproducible builds; no outbound calls by default

3) Architecture Overview (Option 2: Hybrid)

```
Source PDF (German)         Translated PDF (English, via Gemini HTML)
        |                                |
   PDF Parser                       HTML/PDF Parser
        |                                |
  Structured JSON (blocks: headings, paragraphs, tables, lists, images, footnotes, page/bbox)
        -> Alignment Engine (numbering + TF-IDF cosine + heuristics)
           -> Missing/Extra Detection
           -> Table/List/Image alignment
           -> Scoring (METEOR, BLEU, COMET-proxy)
           -> Risk Tagger (severity)
           -> Reporter (HTML + JSON + PDF)

Optional: Gemini Pro used only for semantic alignment/formatting when explicitly enabled and approved.
```

4) Inputs, Assumptions and Edge Cases

Inputs: (1) source.pdf (German), (2) translated.pdf (English) produced from Gemini HTML. Assume text-based PDFs. If scanned/bitmap, OCR is disabled by default (can enable on-prem Tesseract later). Edge cases handled: multi-column layouts, tables with merged cells, nested lists, footnotes, images with captions, rotated pages, headers/footers, long clauses spanning pages, cross-references, and legal numbering gaps.

5) Folder Structure (Enterprise-Ready)

```
translation_comparator/
|-- compare_docs.py                 # CLI entry
|-- config.yaml                     # thresholds, flags, paths
|
|-- extractor/
|   |-- pdf_extractor.py            # pdfplumber/pdfminer extraction incl. tables, bbox, page
|   |-- structure_parser.py         # assemble canonical blocks; numbering; lists; footnotes; captions
|
|-- aligner/
|   |-- align_blocks.py             # TF-IDF + cosine; heading/numbering heuristics; greedy/Hungarian
|   |-- missing_extra.py            # detect and mark missing/extra; penalties; metadata
|   |-- table_align.py              # header detection; row/col/cell alignment; diffs
|
|-- scorer/
|   |-- meteor_scorer.py            # METEOR via NLTK + WordNet
|   |-- bleu_scorer.py              # BLEU via sacrebleu
|   |-- comet_proxy.py              # cosine adequacy proxy (optional)
|
|-- reports/
|   |-- report_json.py              # JSON assembly (sections + document_summary)
|   |-- report_html.py              # Jinja2 two-column bilingual; scores and highlights
|   |-- report_pdf.py               # PDF export via reportlab or wkhtmltopdf shell
|
|-- utils/
|   |-- text_cleaner.py             # unicode normalize, whitespace, hyphenation fixes
|   |-- table_handler.py            # parse/normalize complex tables; merged cells
|   |-- risk_sections.py            # clause taxonomy + severity mapping; boilerplate detection
|   |-- html_sanitizer.py           # sanitize report HTML; safe subset only
|   |-- config_loader.py            # read and validate config.yaml
|
`-- tests/
    |-- test_extractor.py
    |-- test_aligner.py
    |-- test_scorer.py
    |-- test_reports.py
    `-- fixtures/                   # sample PDFs/HTML/JSON goldens
```

6) Module Responsibilities

extractor/pdf_extractor.py

- Use pdfplumber/pdfminer.six to read pages, extract text/spans with bbox, tables, headers/footers hints
- Output raw blocks tagged with type, page, bbox

extractor/structure_parser.py

- Normalize to canonical Block schema; infer section_id and hierarchy; split lists; attach captions and footnotes

aligner/align_blocks.py

- Candidate matches via numbering and header similarity; TF-IDF cosine on cleaned text
- Resolve alignments with greedy or Hungarian algorithm; sliding-window search

aligner/missing_extra.py

- Find source-only (missing_in_target) and target-only (extra_in_target); mark issues; apply score=0.0

aligner/table_align.py

- Match tables by header similarity and section proximity; align rows/columns; detect missing/extra cells

scorer/*

- METEOR (NLTK WordNet), BLEU (sacrebleu), COMET-proxy (cosine adequacy)
- Aggregate to document level with coverage penalty

reports/*

- JSON: sections + summary
- HTML: two columns (German | English); scores; highlights; summary box
- PDF: render auditor copy (reportlab or shell to wkhtmltopdf)

utils/*

- Cleaning, risk taxonomy, HTML sanitization, config loader

7) Canonical Data Schemas

```
// Block
{{
  "id": "src-0051",
  "doc": "source|target",
  "page": 7,
  "bbox": [0,0,0,0],
  "type": "heading|paragraph|table|list_item|image|caption|footnote|header|footer",
  "section_id": "4.3",
  "section_path": ["4","3"],
  "text": "raw text if applicable",
  "table": [["Header A","Header B"],["Val1","Val2"]],
  "list_level": 2,
  "figure_ref": "Figure 2",
  "footnote_ref": "fn-12"
}}
```

```
// SectionComparison
{{
  "section_id": "4.3",
  "source_text": "...",
  "target_text": "...",
  "alignment_status": "aligned|partial_match|missing_in_target|extra_in_target",
  "scores": {{"meteor": 0.83, "bleu": 0.71, "comet_proxy": 0.79}},
  "severity": "high|medium|low",
  "issues": ["terminology mismatch", "numbering discrepancy"]
}}
```

```
// DocumentSummary
{{
  "total_sections_source": 120,
  "total_sections_target": 118,
  "aligned": 112,
  "missing_in_target": 6,
  "extra_in_target": 4,
  "overall_meteor": 0.80,
  "overall_bleu": 0.76,
  "overall_comet_proxy": 0.78,
  "quality_index": 0.77,
  "critical_issues": ["Missing 5.2 Liability exclusions", "Extra clause after 6.2 Confidentiality"]
}}
```

8) Alignment Rules and Algorithms

Order and Hierarchy

- Use section_id and heading similarity first; then TF-IDF cosine on paragraph text
- Sliding-window candidate search to preserve locality; backtracking allowed

Thresholds (configurable)

- High >= 0.80 -> aligned
- 0.60-0.79 -> review (partial_match)
- < 0.60 -> likely mismatch

Tables

- Detect headers; align by header similarity and column index; flag missing/extra rows/cells

Lists

- Preserve nesting; align item-by-item using order and cosine

Images and Captions

- Align by caption text or order; render as placeholders with captions; optional OCR off by default

Headers/Footers

- Parse but mark severity low unless policy promotes them

9) Handling Missing and Extra Sections

```
// Missing in Target
{{
  "alignment_status": "missing_in_target",
  "target_text": null,
  "scores": {{"meteor": 0.0, "bleu": 0.0, "comet_proxy": 0.0}},
  "issues": ["Entire section missing in translation"]
}}

// Extra in Target
{{
  "alignment_status": "extra_in_target",
  "source_text": null,
  "scores": {{"meteor": 0.0, "bleu": 0.0, "comet_proxy": 0.0}},
  "issues": ["Extra section found in translation"]
}}
```

Coverage penalty:

```
coverage = aligned_sections / total_sections_source
QI_final = QI_base * coverage
```

10) Scoring and Aggregation

Per-Section:

- METEOR via NLTK+WordNet (English side; German source compared post-translation pairing)
- BLEU via sacrebleu for n-gram overlap
- COMET-proxy: cosine similarity between vectorized texts (TF-IDF); if in-house embeddings available, swap

Aggregation (length-weighted by source tokens):

```
overall_meteor = sum(len(src)*meteor)/sum(len(src))
overall_bleu   = sum(len(src)*bleu)  /sum(len(src))
overall_comet  = sum(len(src)*comet) /sum(len(src))
```

Quality Index (configurable weights in config.yaml):

```
QI_base  = 0.5*overall_meteor + 0.3*overall_comet + 0.2*overall_bleu
QI_final = QI_base * (aligned_sections / total_sections_source)
```

11) Risk Tagging and Severity Model

High Severity

- Liability, Indemnity, Limitation of Liability, Termination, Governing Law, Jurisdiction, Fees/Interest, Security/Collateral

Medium

- Confidentiality, Data Protection, Audit Rights, Service Levels, Force Majeure

Low

- Formatting, headers/footers, pagination

Rules

- Missing/Extra in high-severity -> critical_issues
- Terminology mismatches in key terms increase severity
- Boilerplate detection: tag as "possible_boilerplate" (still extra_in_target but medium severity)

12) Reporting - HTML and PDF

HTML (Jinja2 two-column, left=German, right=English):

- Show section headers bold; paragraphs aligned
- Scores under each block (METEOR | BLEU | COMET-proxy)
- Highlights: green=aligned, yellow=review, red=missing/extra

- Top summary box: overall scores, Quality Index, Critical Issues
PDF:

- reportlab direct rendering, or wkhtmltopdf (ops-approved) from HTML

13) CLI, Config and Thresholds

CLI

```
python compare_docs.py source.pdf translated.pdf --out report.html --pdf --config config.yaml
```

config.yaml

```
thresholds:
  high: 0.80
  review: 0.60
penalties:
  missing_extra_zero: true
weights:
  meteor: 0.5
  comet_proxy: 0.3
  bleu: 0.2
risk_keywords:
  high: ["liability","indemnity","jurisdiction","governing law","termination","interest","collateral"]
  medium: ["confidentiality","data protection","service level","audit"]
render:
  engine: "jinja2"
  pdf_export: "reportlab"   # or "wkhtmltopdf"
security:
  outbound_network: false
  pii_log_truncation: 256
```

14) Logging, Audit and Security

Logging:

- Per block: similarity scores, decision, thresholds, status (aligned/missing/extra)
- Input digests (SHA-256), library versions, config hash
- PII-safe logs: truncate long texts; never persist full docs in logs
Audit Artifacts:

- report.json, report.html, report.pdf
- alignment_trace.json (optional): step-by-step matching rationale
Security:

- No external API calls; on-prem only; dependency pinning; SBOM recommended
- Role-based access for sensitive outputs; redaction option for exports

15) Performance and Scalability

- Stream parsing; avoid loading entire PDF in memory
- Reuse TF-IDF vectorizers by section; batch vectorization
- Sliding-window alignment to avoid O(n^2) across entire doc
- Optional multiprocessing per page/section (respect GIL by batching work)
- Memory guardrails and backpressure for 500+ page docs

16) Testing Strategy

Unit:

- extractor: bbox and order, tables, lists, footnotes
- aligner: numbering heuristics, threshold behaviors, missing/extra tagging
- scorer: metric sanity ranges, zeroing on missing/extra

- reports: HTML validity, PDF generation smoke
Integration:

- Golden fixtures (fixtures/) with expected JSON and HTML snapshot tests
Regression:

- Snapshot compare of JSON outputs; protect thresholds via tests

17) Gemini Prompt Pack (Used Between Parser and Reporter)

System Prompt

```
You are a document alignment and comparison engine for bank-grade legal documents. Keep structure (sections,
align paragraphs/tables/images, and output valid HTML + JSON with scores and issues. Do not summarize or omi
```

User Inputs

```
- Source JSON (German) - exact content and order
- Target JSON (English) - exact content and order
```

Alignment and Scoring Prompt

```
Align block-by-block (section->paragraph->table->figure). For each pair, output:
- alignment_status: perfect_match|partial_match|missing_in_target|extra_in_target
- comet_score (0-1, proxy or 0.0), meteor_score (0-1)
- issues: []
After all sections, compute document_summary with length-weighted overall scores, coverage penalty, critical
and a single quality_index.
Generate a two-column HTML report (left German, right English) with scores under each block and red highligh
for missing/extra. Output only the JSON and HTML.
```

18) Sample CLI Driver (Pseudocode)

```
def main(src_pdf, tgt_pdf, out_html, out_pdf, cfg):
    src_blocks = extractor.pdf_extractor.extract(src_pdf)
    tgt_blocks = extractor.pdf_extractor.extract(tgt_pdf)

    src_struct = extractor.structure_parser.to_blocks(src_blocks)
    tgt_struct = extractor.structure_parser.to_blocks(tgt_blocks)

    pairs = aligner.align_blocks.match(src_struct, tgt_struct, cfg)
    pairs = aligner.missing_extra.annotate(pairs, src_struct, tgt_struct, cfg)
    pairs = aligner.table_align.align_tables(pairs, cfg)

    per_section = scorer.compute_all(pairs, cfg)  # meteor, bleu, comet_proxy per section
    summary = scorer.aggregate(per_section, pairs, cfg)  # overall scores + QI + critical_issues

    reports.report_json.write(per_section, summary, "report.json")
    reports.report_html.write(per_section, summary, out_html, cfg)
    if out_pdf:
        reports.report_pdf.write(out_html, out_pdf, cfg)
```

19) Deployment, Governance and Roadmap

Deployment:

- Containerize; pin dependencies; produce SBOM; disable outbound network by default
- Secrets-free runtime; all artifacts stored on approved volumes
Governance:

- Change control with versioned config and thresholds
- Model/library approval gates; continuous compliance scans
Roadmap:

- In-house embeddings for stronger alignment
- On-prem OCR for images/tables (Tesseract)
- Reviewer UI for triage and corrections

- Exact layout mirroring using page/bbox in report renderer

20) Final Notes

This specification consolidates the entire discussion: enterprise libraries, hybrid architecture, code structure, schemas, alignment rules, scoring with penalties, missing/extra handling, severity tagging, prompts, CLI/config, reporting, logging/audit, performance, testing, deployment, and roadmap. Ready for engineers to implement in Cursor.