

Konversi PDF ke Corpus Siap Upload ke RAG LLM

Onno W. Purbo
onno@indo.net.id
onno@itts.ac.id
Twitter onnowpurbo

**Institut Teknologi Tangerang Selatan
Dinas KOMINFO Tangerang Selatan
OnnoCenter
2026**

Lisensi: Karya ini dilisensikan di bawah **Creative Commons Attribution (CC BY) 4.0 International**. Anda bebas menggunakan, membagikan, dan mengadaptasi karya ini untuk tujuan apa pun, termasuk komersial, dengan tetap mencantumkan atribusi yang layak kepada penulis.

Disclaimer: Buku ini disusun untuk tujuan edukasi, dokumentasi, dan refleksi publik berdasarkan sumber-sumber yang dianggap kredibel pada saat penulisan. Penulis telah berupaya menjaga akurasi, namun sangat mungkin masih terdapat kekurangan atau kekeliruan; karena itu, buku ini tidak dimaksudkan sebagai satu-satunya rujukan yang final, dan penulis terbuka terhadap koreksi serta masukan yang membangun

PDF tidak boleh langsung “dilempar” ke RAG; PDF harus diubah dulu menjadi corpus bersih, berstruktur, ber-metadata, ter-chunk dengan baik, dan memiliki jejak sumber agar LLM dapat mencari, menjawab, dan mempertanggungjawabkan jawabannya.

Kata Pengantar

Buku ini disusun sebagai panduan praktis untuk membantu mahasiswa, peneliti, dan praktisi mengubah dokumen PDF menjadi **corpus yang bersih, terstruktur, ber-metadata, dan siap digunakan dalam sistem RAG LLM**. Fokus utama buku ini bukan sekadar mengekstrak teks dari PDF, tetapi membangun alur kerja yang rapi: mulai dari audit kualitas PDF, OCR, konversi ke Markdown, pembersihan teks, *chunking*, metadata, ekstraksi pengetahuan, hingga persiapan file untuk diunggah ke *Knowledge Base*.

Melalui pendekatan *open source* dan praktik bertahap, pembaca diajak memahami bahwa kualitas jawaban AI sangat ditentukan oleh kualitas corpus yang diberikan. PDF yang indah bagi manusia belum tentu siap dibaca oleh mesin. Karena itu, buku ini menekankan disiplin teknis, dokumentasi, validasi, dan keterlacakan sumber agar penggunaan LLM menjadi lebih akurat, bertanggung jawab, dan dapat diaudit.

Semoga buku ini menjadi bekal awal bagi pembaca untuk membangun sistem RAG yang lebih kuat, mandiri, dan bermanfaat.

Jakarta, Mei 2026

Onno W. Purbo

Executive Summary

Buku “**Konversi PDF ke Corpus Siap Upload ke RAG LLM**” menegaskan satu pesan strategis: masa depan pemanfaatan LLM tidak ditentukan hanya oleh besarnya model, tetapi oleh **kualitas corpus yang masuk ke dalam sistem RAG**. PDF mentah tidak boleh langsung diunggah karena sering membawa masalah tersembunyi: teks hasil scan, urutan baca kacau, tabel rusak, *header-footer* berulang, *OCR noise*, metadata hilang, dan potongan informasi yang tidak dapat dilacak sumbernya. Buku ini membangun cara berpikir baru bagi mahasiswa, peneliti, dan praktisi: dari sekadar “mengunggah dokumen” menjadi **merekayasa pengetahuan**. Alurnya jelas dan praktis: audit kualitas PDF, OCR, ekstraksi ke teks dan Markdown, pembersihan corpus, normalisasi struktur, penambahan metadata, *chunking*, pembuatan JSONL/CSV, ekstraksi entitas dan relasi, pembangunan ontologi ringan, lalu upload ke *Knowledge Base* RAG. Kekuatan buku ini terletak pada disiplin teknis yang sederhana namun kritis: setiap chunk harus bersih, cukup konteks, memiliki identitas, halaman sumber, bagian dokumen, dan jejak audit. Dengan pendekatan *open source*, Python, OCRmyPDF, Tesseract, Docling, PyMuPDF, pdfplumber, spaCy, RDF, dan Open WebUI, buku ini tidak hanya mengajarkan alat, tetapi membangun budaya kerja yang dapat diulang, diperiksa, dan dipertanggungjawabkan. Pesan inspirasionalnya tajam: **RAG yang kuat dimulai bukan dari model yang besar, tetapi dari corpus yang benar**. Buku ini penting karena menggeser AI dari sekadar mesin jawaban menjadi sistem pengetahuan yang lebih akurat, transparan, dapat diaudit, dan relevan untuk pendidikan, riset, organisasi, serta pengembangan AI lokal yang mandiri.

Daftar isi

Kata Pengantar.....	4
Executive Summary.....	5
Daftar isi.....	6
BAB 1 — Mengapa PDF Harus Diubah Menjadi Corpus RAG.....	15
1.1 Masalah Besar di Balik PDF.....	15
1.2 Masalah PDF untuk AI.....	16
1.3 Bedanya PDF, Teks, Corpus, Knowledge Base, Vector Database, Knowledge Graph, dan Ontologi.....	17
1.4 Mengapa Upload PDF Mentah Sering Menghasilkan Jawaban Buruk.....	18
1.5 Prinsip RAG: Retrieve First, Generate Later.....	19
1.6 Corpus RAG yang Baik.....	20
1. Bersih.....	20
2. Terstruktur.....	20
3. Ber-metadata.....	20
4. Terpotong rapi.....	20
5. Bisa dilacak.....	20
6. Memiliki knowledge layer.....	20
1.7 Mengapa Metadata Sangat Penting.....	21
1.8 Mengapa Ontologi Ringan Membantu Corpus RAG.....	22
1.9 Pipeline Besar Buku.....	23
1.10 Studi Mini: PDF Panduan Akademik.....	24
Praktik Mini Bab 1.....	25
Tujuan Praktik.....	25
File 1 — project_plan.md.....	25
File 2 — sample_pdf_list.csv.....	26
Ringkasan Bab 1.....	26
BAB 2 — Menyiapkan Workspace Python dan Tool Open Source.....	27
2.1 Dari Ide ke Lingkungan Kerja.....	27
2.2 Prinsip Kerja: Lokal, Terbuka, dan Bisa Diulang.....	28
1. Lokal.....	28
2. Open Source First.....	28
3. Bisa Diulang.....	28
2.3 Struktur Folder Proyek.....	29
Fungsi Setiap Folder.....	29
2.4 Membuat Folder Proyek.....	30
Penjelasan Perintah.....	30
2.5 Instalasi Python dan Virtual Environment.....	31
Membuat Virtual Environment.....	31
Penjelasan Perintah.....	31
2.6 Instalasi OCRmyPDF dan Tesseract.....	32
Instalasi di Ubuntu / Debian.....	32
Penjelasan Perintah.....	32

Tes OCRmyPDF.....	32
2.7 Instalasi Library Python Utama.....	33
Fungsi Library.....	33
Instal Model Bahasa spaCy.....	33
2.8 Instalasi dan Penggunaan Docling.....	34
Instalasi Docling.....	34
Tes Docling.....	34
Contoh Konversi PDF ke Markdown.....	34
2.9 Pengenalan Open WebUI Knowledge Base.....	35
File yang Akan Disiapkan untuk Upload.....	35
2.10 Membuat File Requirements.....	36
Penjelasan Perintah.....	36
2.11 Script Cek Environment.....	37
Penjelasan Ringkas Script.....	38
2.12 Membuat File workspace_ready.md.....	39
2.13 Praktik Utama Bab 2.....	40
Tugas Praktik.....	40
Output Wajib.....	40
Kriteria Berhasil.....	40
2.14 Troubleshooting Ringkas.....	41
Ringkasan Bab 2.....	41
BAB 3 — Audit Kualitas PDF sebelum Diproses.....	42
3.1 Audit Dulu, Proses Kemudian.....	42
3.2 PDF Searchable vs PDF Scan.....	42
1. PDF Searchable.....	42
2. PDF Scan.....	43
3.3 PDF Dua Kolom.....	43
3.4 PDF dengan Tabel.....	44
3.5 PDF dengan Gambar dan Diagram.....	45
3.6 PDF dengan Halaman Miring, Buram, atau Pecah.....	46
3.7 Mengecek Jumlah Halaman dan Panjang Teks per Halaman.....	47
3.8 Menentukan Apakah PDF Perlu OCR.....	48
3.9 Praktik Python Bab 3.....	49
Tujuan Praktik.....	49
Script: scripts/audit_pdf_quality.py.....	49
3.10 Penjelasan Script.....	51
Bagian 1 — Import Library.....	51
Bagian 2 — Lokasi Folder.....	51
Bagian 3 — Ambang Deteksi Scan.....	51
Bagian 4 — Ekstraksi Panjang Teks per Halaman.....	51
Bagian 5 — Audit Satu PDF.....	52
Bagian 6 — Menyimpan CSV.....	52
3.11 Menjalankan Script.....	52
3.12 Cara Membaca Hasil Audit.....	53

Kasus 1 — PDF Baik.....	53
Kasus 2 — PDF Scan.....	53
Kasus 3 — PDF Meragukan.....	53
3.13 Checklist Audit Manual.....	54
3.14 Status Keputusan Setelah Audit.....	54
Output Bab 3.....	55
Ringkasan Bab 3.....	55
BAB 4 — Ekstraksi PDF ke Teks dan Markdown.....	56
4.1 Bab Inti Teknis Pertama.....	56
4.2 Mengapa Perlu Banyak Pipeline.....	56
4.3 Prinsip Ekstraksi untuk Corpus RAG.....	57
4.4 Persiapan Folder.....	58
4.5 Pipeline A — Ekstraksi PDF Searchable dengan PyMuPDF.....	59
Tujuan.....	59
Script: scripts/extract_with_pymupdf.py.....	59
Penjelasan Script.....	60
4.6 Pipeline B — OCR PDF Scan dengan OCRmyPDF.....	61
Tujuan.....	61
Perintah OCR.....	61
Penjelasan Perintah.....	61
Ekstrak Hasil OCR dengan PyMuPDF.....	62
4.7 Pipeline C — Konversi PDF Kompleks ke Markdown dengan Docling.....	63
Tujuan.....	63
Perintah Docling.....	63
Catatan Praktis.....	63
4.8 Ekstraksi Teks dan Tabel dengan pdfplumber.....	64
Tujuan.....	64
Script: scripts/extract_with_pdfplumber.py.....	64
Penjelasan Script.....	65
4.9 Menjaga Heading, Paragraf, Tabel, dan Urutan Baca.....	66
Heading.....	66
Paragraf.....	66
Tabel.....	66
Urutan Baca.....	67
4.10 Membandingkan Hasil Ekstraksi Antar-Tool.....	68
Kriteria Perbandingan.....	68
Script: scripts/compare_extraction_outputs.py.....	68
4.11 Menentukan Hasil Terbaik untuk Corpus RAG.....	70
Rekomendasi Keputusan.....	70
Aturan Praktis.....	70
4.12 Menyimpan Hasil Mentah dan Markdown.....	71
4.13 Praktik Utama Bab 4.....	72
Tugas Praktik.....	72
Output Wajib.....	72

Output Tambahan.....	72
4.14 Checklist Validasi Hasil Ekstraksi.....	73
4.15 Troubleshooting Ringkas.....	73
4.16 Ringkasan Bab 4.....	74
BAB 5 — Cleaning, Normalisasi, dan Metadata Corpus.....	75
5.1 Dari Teks Mentah ke Corpus Bersih.....	75
5.2 Peta Halaman Bab 5.....	75
5.3 Raw Text vs Clean Corpus.....	76
5.4 Menghapus Header, Footer, Nomor Halaman, dan Watermark.....	77
5.5 Noise OCR yang Sering Muncul.....	78
5.6 Normalisasi Spasi, Tanda Baca, dan Paragraf.....	79
5.7 Menjaga Struktur Dokumen agar Mudah di-Chunk.....	80
5.8 Metadata Dokumen.....	81
5.9 YAML Front Matter untuk Markdown.....	82
5.10 Corpus Index.....	83
5.11 Praktik Python: Script Cleaning Corpus.....	84
Tujuan Praktik.....	84
Script: scripts/clean_markdown_corpus.py.....	84
5.12 Penjelasan Script.....	87
Bagian 1 — Import Library.....	87
Bagian 2 — Metadata Dokumen.....	87
Bagian 3 — Lokasi File.....	88
Bagian 4 — Menghapus Page Marker dan Nomor Halaman.....	88
Bagian 5 — Menghapus Header Berulang.....	88
Bagian 6 — Normalisasi Spasi.....	89
Bagian 7 — Koreksi OCR Umum.....	89
Bagian 8 — Membuat YAML Front Matter.....	90
Bagian 9 — Membuat Corpus Index.....	90
5.13 Menjalankan Script.....	91
5.14 Contoh Output Clean Corpus.....	92
5.15 Contoh Output Corpus Index.....	93
5.16 Versioning Corpus dengan Git.....	93
5.17 Checklist Validasi Cleaning.....	93
5.18 Troubleshooting Ringkas.....	94
5.19 Praktik Utama Bab 5.....	95
Tugas Praktik.....	95
Output Wajib.....	95
Output Tambahan.....	95
Ringkasan Bab 5.....	96
BAB 6 — Chunking Corpus agar Mudah Dicari LLM.....	97
6.1 Pembuka Bab — Mengapa Chunking Menentukan Kualitas RAG.....	97
6.2 Peta Halaman Bab 6.....	97
6.3 Apa Itu Chunk dalam RAG.....	98
6.4 Chunk Terlalu Kecil vs Chunk Terlalu Besar.....	99

Chunk Terlalu Kecil.....	99
Chunk Terlalu Besar.....	99
6.5 Strategi Chunk Berbasis Heading.....	100
6.6 Strategi Chunk Berbasis Paragraf.....	101
6.7 Strategi Chunk Berbasis Halaman.....	102
6.8 Chunk Overlap.....	103
6.9 Metadata per Chunk.....	104
6.10 Format JSONL untuk Chunk.....	105
6.11 Praktik Python: Membuat Chunk dari Clean Markdown.....	106
Tujuan Praktik.....	106
Script: scripts/create_chunks.py.....	106
6.12 Penjelasan Script.....	110
Import Library.....	110
Parameter Chunk.....	110
Menghapus YAML Front Matter.....	110
Split Berdasarkan Heading.....	111
Membagi Teks Panjang.....	111
Menyimpan JSONL.....	111
6.13 Menjalankan Script.....	112
6.14 Contoh Output chunks.jsonl.....	112
6.15 Contoh Output chunk_report.csv.....	113
6.16 Evaluasi Chunk: Apakah Masih Bisa Menjawab Pertanyaan?.....	114
6.17 Membuat Folder rag_upload/.....	115
6.18 Checklist Validasi Chunk.....	116
6.19 Troubleshooting Ringkas.....	116
6.20 Praktik Utama Bab 6.....	117
Tugas Praktik.....	117
Output Wajib.....	117
Output Tambahan.....	117
Ringkasan Bab 6.....	117
BAB 7 — Menyiapkan File Upload RAG: Markdown, JSONL, dan CSV.....	118
7.1 Pembuka Bab — Dari File Kerja ke File Final.....	118
7.2 Peta Halaman Bab 7.....	119
7.3 File Kerja Internal vs File Upload RAG.....	119
7.4 Format File yang Umum Dipakai untuk RAG.....	120
7.5 Markdown untuk Dokumen Bersih.....	121
7.6 JSONL untuk Chunk Terstruktur.....	122
7.7 CSV untuk Metadata dan Evaluasi.....	122
7.8 Struktur Folder rag_upload/.....	123
7.9 Praktik Python: Membuat Paket Upload RAG.....	123
Tujuan Praktik.....	123
Script: scripts/prepare_rag_upload.py.....	124
7.10 Penjelasan Script.....	127
Bagian 1 — Input dan Output.....	127

Bagian 2 — Validasi File Ada.....	127
Bagian 3 — Validasi JSONL.....	127
Bagian 4 — Membuat README Upload.....	128
Bagian 5 — Menyalin File Final.....	128
Bagian 6 — Membuat Metadata Upload.....	128
7.11 Menjalankan Script.....	129
7.12 Contoh corpus_for_rag.md.....	130
7.13 Contoh corpus_all.jsonl.....	131
7.14 Contoh metadata.csv.....	131
7.15 Validasi Kesiapan Upload.....	132
Checklist Markdown.....	132
Checklist JSONL.....	132
Checklist CSV.....	132
7.16 Catatan Upload ke Knowledge Base.....	133
7.17 Troubleshooting Ringkas.....	133
7.18 Praktik Utama Bab 7.....	134
Tugas Praktik.....	134
Output Wajib.....	134
Ringkasan Bab 7.....	134
BAB 8 — Ekstraksi Knowledge: Entitas, Konsep, Relasi, dan Evidence.....	135
8.1 Pembuka Bab — Dari Corpus ke Knowledge Layer.....	135
8.2 Peta Halaman Bab 8.....	136
8.3 Apa Itu Entitas, Konsep, Istilah, Relasi, dan Evidence.....	136
8.4 Mengapa Evidence Wajib Disimpan.....	137
8.5 Ekstraksi Keyword Sederhana dengan Python.....	138
8.6 Ekstraksi Entitas dengan spaCy / Stanza.....	139
8.7 Ekstraksi Konsep dari Heading dan Istilah Berulang.....	140
8.8 Ekstraksi Relasi Berbasis Pola Kalimat.....	141
8.9 Ekstraksi Relasi Berbasis Prompt LLM Lokal.....	142
8.10 Mencegah Halusinasi Knowledge Extraction.....	143
8.11 Praktik Python: Ekstraksi Entitas, Konsep, Relasi, dan Evidence.....	144
Tujuan Praktik.....	144
Script: scripts/extract_knowledge.py.....	144
8.12 Penjelasan Script.....	150
Input.....	150
Domain Entities.....	150
Ekstraksi Entitas.....	150
Ekstraksi Konsep.....	151
Ekstraksi Relasi.....	151
Evidence Table.....	151
8.13 Menjalankan Script.....	152
8.14 Contoh Output entities.csv.....	153
8.15 Contoh Output concepts.csv.....	153
8.16 Contoh Output relations.csv.....	154

8.17 Contoh Output evidence_table.csv.....	154
8.18 Validasi Manual.....	155
8.19 Menghubungkan ke Ontologi Ringan.....	156
8.20 Praktik Utama Bab 8.....	156
Tugas Praktik.....	156
Output Wajib.....	156
8.21 Troubleshooting Ringkas.....	157
Ringkasan Bab 8.....	157
BAB 9 — Ontologi Ringan untuk Memperkaya Corpus RAG.....	158
9.1 Ontologi sebagai Peta Pengetahuan.....	158
9.2 Peta Halaman Bab 9.....	159
9.3 Ontologi Ringan untuk RAG, Bukan Ontologi Akademik yang Rumit.....	159
9.4 Taxonomy vs Knowledge Graph vs Ontology.....	160
9.5 Class, Instance, Property, dan Relation.....	161
Class.....	161
Instance.....	161
Property / Relation.....	161
9.6 RDF Triple: Subject–Predicate–Object.....	162
9.7 Turtle: Format Ringkas untuk RDF.....	163
9.8 Praktik Python: Mengubah relations.csv menjadi Turtle.....	164
Tujuan Praktik.....	164
Script: scripts/build_lightweight_ontology.py.....	164
9.9 Penjelasan Script.....	167
Library Utama.....	167
Fungsi slugify().....	167
Membuat Class.....	167
Membuat Individual.....	167
Membuat Object Property.....	167
Menambahkan Evidence.....	168
9.10 Contoh Output Turtle.....	168
9.11 Membuat Diagram Mermaid dari Relasi.....	169
9.12 Membuka Hasil di Protégé.....	171
9.13 Menggunakan Ontologi sebagai Metadata Pencarian RAG.....	172
9.14 Validasi Ontologi Ringan.....	173
9.15 Praktik Validasi RDF dengan Python.....	174
9.16 SPARQL Sekilas.....	175
9.17 Output Bab 9.....	175
9.18 Troubleshooting Ringkas.....	176
9.19 Praktik Utama Bab 9.....	176
Ringkasan Bab 9.....	177
BAB 10 — Studi Kasus End-to-End: PDF ke Corpus Siap Upload RAG.....	178
10.1 Proyek Akhir Mini.....	178
10.2 Peta Halaman Bab 10.....	179
10.3 Case Practical: Panduan Akademik / SOP Kampus.....	179

10.4 Struktur Folder Akhir.....	180
10.5 Alur Praktik End-to-End.....	181
Langkah 1 — Ambil PDF Sumber.....	181
Langkah 2 — Audit Kualitas PDF.....	181
Langkah 3 — OCR Jika PDF Scan.....	181
Langkah 4 — Ekstrak ke Markdown.....	181
Langkah 5 — Bersihkan Teks dan Tambahkan Metadata.....	182
Langkah 6 — Buat Chunk.....	182
10.6 Membuat File Upload RAG.....	183
10.7 Ekstrak Entitas dan Relasi.....	184
10.8 Buat Ontologi Ringan.....	185
10.9 Upload ke Open WebUI Knowledge Base.....	186
10.10 Membuat 10 Pertanyaan Uji.....	187
10.11 Prompt Uji di Open WebUI.....	188
10.12 Mencatat Hasil Evaluasi.....	189
10.13 Script Membuat Template Evaluasi.....	190
10.14 Script Mengecek Kelengkapan Final Project.....	191
10.15 Cara Merevisi Corpus Berdasarkan Hasil Uji.....	192
Masalah 1 — Jawaban Tidak Ditemukan.....	192
Masalah 2 — Jawaban Salah.....	192
Masalah 3 — Jawaban Terlalu Panjang.....	193
10.16 Kriteria Keberhasilan Proyek.....	193
10.17 Output Akhir Proyek.....	194
10.18 Template revision_notes.md.....	195
Ringkasan Bab 10.....	195
Lampiran A — Checklist PDF ke Corpus RAG.....	196
Lampiran B — Template Metadata Corpus.....	198
Kolom Wajib.....	198
Contoh Metadata Corpus.....	198
Penjelasan Kolom.....	199
Status yang Disarankan.....	199
Lampiran C — Template Chunk RAG.....	200
Kolom Wajib.....	200
Contoh CSV Chunk.....	200
Contoh JSONL Chunk.....	200
Penjelasan Kolom.....	201
Checklist Chunk.....	201
Lampiran D — Template Entity dan Relation Table.....	202
Kolom Wajib.....	202
Contoh Entity dan Relation Table.....	202
Penjelasan Kolom.....	202
Skala Confidence.....	203
Aturan Validasi.....	203
Lampiran E — 30 Prompt AI untuk Membuat Corpus RAG.....	204

A. Prompt Cek Kualitas Teks.....	204
B. Prompt Ekstraksi Konsep.....	206
C. Prompt Ekstraksi Entitas.....	207
D. Prompt Ekstraksi Relasi.....	209
E. Prompt Pembuatan Metadata.....	211
F. Prompt Validasi Chunk.....	212
G. Prompt Pembuatan Pertanyaan Uji.....	213
H. Prompt Evaluasi Jawaban RAG.....	214
Lampiran F — Script Python Ringkas.....	216
F.1 Struktur Folder yang Dipakai.....	216
F.2 Instalasi Library Python.....	217
F.3 Script 1 — Cek PDF.....	218
F.4 Script 2 — OCR Batch.....	220
F.5 Script 3 — Ekstraksi Teks.....	222
F.6 Script 4 — Konversi Markdown dengan Docling.....	224
F.7 Script 5 — Cleaning.....	226
F.8 Script 6 — Chunking.....	229
F.9 Script 7 — Entity Extraction.....	233
F.10 Script 8 — Relation Extraction.....	235
F.11 Script 9 — CSV ke RDF/Turtle.....	239
F.12 Script 10 — Jalankan Semua Pipeline.....	242
F.13 Checklist Setelah Menjalankan Script.....	244
Catatan Penutup Lampiran F.....	244
Glossary.....	245
Referensi.....	250
Tentang Penulis.....	253

BAB 1 — Mengapa PDF Harus Diubah Menjadi Corpus RAG

LLM tidak membutuhkan PDF yang indah dilihat manusia; LLM membutuhkan corpus yang bersih, terstruktur, ber-metadata, terpotong rapi, dan dapat ditelusuri kembali ke sumber aslinya.

1.1 Masalah Besar di Balik PDF

PDF adalah format dokumen yang sangat populer karena rapi, stabil, dan mudah dibagikan. Namun, untuk AI dan LLM, PDF sering menjadi **format yang sulit dipahami**. Masalahnya bukan hanya isi dokumen, tetapi juga cara isi itu tersimpan: teks bisa terpecah, urutan baca bisa kacau, tabel bisa rusak, halaman scan bisa tidak memiliki teks, dan header-footer bisa ikut terbaca sebagai isi utama.

Dalam sistem *Retrieval-Augmented Generation* atau RAG, kualitas jawaban LLM sangat bergantung pada kualitas bahan yang dicari. Jika bahan yang masuk buruk, hasil pencarian juga buruk. Jika hasil pencarian buruk, jawaban LLM mudah menjadi tidak lengkap, salah konteks, atau bahkan halusinatif.

Karena itu, buku ini tidak mengajarkan mahasiswa untuk sekadar **mengunggah PDF mentah**. Buku ini mengajarkan cara mengubah PDF menjadi **corpus RAG**: kumpulan teks bersih, terstruktur, ber-metadata, dan siap dipakai oleh sistem pencarian LLM.

Dokumen outline buku yang digunakan sebagai arah utama menegaskan bahwa target akhir buku adalah membuat **file corpus siap upload ke RAG**, dengan format utama seperti Markdown, JSONL, CSV metadata, entity table, relation table, dan ontologi ringan.

1.2 Masalah PDF untuk AI

PDF dirancang terutama untuk **tampilan dokumen**, bukan untuk menjadi bahan pengetahuan AI. Manusia dapat melihat judul, tabel, catatan kaki, kolom, dan gambar secara visual. LLM tidak bekerja seperti itu. LLM membaca teks yang sudah diekstrak, lalu sistem RAG mencari potongan teks yang dianggap relevan.

Masalah umum PDF untuk AI:

- **PDF scan** sering hanya berisi gambar, bukan teks.
- **Layout dua kolom** dapat membuat urutan bacaan menjadi kacau.
- **Tabel** sering berubah menjadi teks berantakan.
- **Header dan footer** dapat terulang di setiap halaman dan mengganggu pencarian.
- **Nomor halaman, watermark, dan catatan kaki** bisa masuk sebagai noise.
- **OCR noise** dapat mengubah kata, angka, atau istilah teknis.
- **Gambar dan diagram** sering hilang maknanya jika tidak diberi keterangan.
- **Struktur heading** bisa hilang, padahal heading penting untuk konteks.

OCRmyPDF relevan untuk masalah PDF scan karena dokumentasi resminya menyatakan bahwa tool ini menambahkan lapisan teks OCR ke PDF scan sehingga dokumen dapat dicari. ([OCRmyPDF](#)) Docling juga relevan karena dokumentasi resminya menyebut kemampuan parsing dokumen, termasuk pemahaman PDF tingkat lanjut, dan integrasi dengan ekosistem *generative AI*. ([Docling Project](#))

Inti kritiknya sederhana: PDF yang bagus untuk mata manusia belum tentu bagus untuk RAG.

1.3 Bedanya PDF, Teks, Corpus, Knowledge Base, Vector Database, Knowledge Graph, dan Ontologi

Mahasiswa perlu membedakan istilah-istilah dasar sejak awal agar tidak salah memahami workflow RAG.

Istilah	Makna Praktis
PDF	File dokumen akhir untuk dibaca manusia. Bisa berisi teks, gambar, tabel, scan, atau campuran semuanya.
Teks	Hasil ekstraksi dari PDF. Belum tentu bersih atau benar urutannya.
Corpus	Kumpulan teks yang sudah disusun, dibersihkan, diberi struktur, dan siap diproses.
Knowledge Base	Tempat penyimpanan pengetahuan yang dapat dicari oleh sistem AI.
Vector Database	Tempat menyimpan representasi numerik dari teks agar dapat dicari berdasarkan kemiripan makna.
Knowledge Graph	Representasi pengetahuan sebagai node dan relasi, misalnya “Mahasiswa — mengisi — KRS”.
Ontologi	Struktur pengetahuan yang lebih formal: konsep, kelas, relasi, properti, dan aturan makna.

Keet menjelaskan bahwa ontology engineering memberi mahasiswa pemahaman tentang ontologi, *knowledge base*, OWL, reasoning, serta pendekatan pengembangan *top-down* dan *bottom-up*, termasuk dari sumber non-ontologis seperti bahasa alami dan basis data. Dalam buku ini, ontologi tidak diposisikan sebagai teori berat, tetapi sebagai **peta pengetahuan ringan** untuk membantu corpus RAG lebih mudah dicari dan diuji.

1.4 Mengapa Upload PDF Mentah Sering Menghasilkan Jawaban Buruk

Mengunggah PDF mentah ke sistem RAG terlihat mudah, tetapi sering menghasilkan masalah. Sistem bisa gagal menemukan jawaban bukan karena LLM bodoh, tetapi karena bahan yang dicari tidak siap.

Contoh masalah:

- Pertanyaan: “**Siapa yang menyetujui KRS mahasiswa?**”
- PDF asli berisi kalimat: “KRS disetujui oleh Dosen Wali.”
- Namun hasil ekstraksi menjadi: “KRS disetu jui oleh Dosen Wa li.”
- Akibatnya, sistem pencarian mungkin gagal mengambil potongan yang tepat.

Masalah lain:

- Tabel syarat layanan berubah menjadi paragraf acak.
- Judul bab hilang sehingga chunk kehilangan konteks.
- Nomor halaman bercampur dengan isi utama.
- Teks dari dua kolom terbaca menyilang.
- Potongan chunk terlalu pendek sehingga makna hilang.
- Potongan chunk terlalu panjang sehingga pencarian tidak presisi.

Open WebUI mendukung pembuatan *Knowledge Base* dengan upload file atau dokumen, lalu knowledge dapat dilampirkan ke model atau direferensikan dalam chat dengan #. ([Open WebUI](#)) Namun, kemampuan upload ini tidak menghilangkan kebutuhan untuk menyiapkan corpus yang rapi. Semakin bersih corpus, semakin besar peluang RAG mengambil konteks yang tepat.

Kesalahan paling umum: menganggap RAG dapat memperbaiki dokumen buruk secara otomatis. Padahal, RAG hanya mencari dari bahan yang tersedia.

1.5 Prinsip RAG: Retrieve First, Generate Later

RAG bekerja dengan prinsip sederhana: **cari dulu, jawab kemudian**.

Urutan dasarnya:

1. Pengguna bertanya.
2. Sistem mencari potongan dokumen yang relevan.
3. Potongan dokumen dikirim ke LLM.
4. LLM menyusun jawaban berdasarkan potongan tersebut.
5. Jawaban idealnya dapat dilacak ke sumber.

Open WebUI menjelaskan fitur *Knowledge & RAG* sebagai cara memberi AI akses ke dokumen agar AI dapat menemukan informasi yang dibutuhkan; tersedia pilihan *vector search* untuk koleksi besar dan *full-content injection* untuk presisi. ([Open WebUI](#))

Dengan prinsip ini, kualitas RAG sangat bergantung pada tiga hal:

- **kualitas teks,**
- **kualitas chunk,**
- **kualitas metadata.**

Jika tiga hal ini buruk, LLM akan menerima konteks yang buruk. Jika konteks buruk, jawaban juga mudah buruk.

1.6 Corpus RAG yang Baik

Corpus RAG yang baik memiliki ciri berikut.

1. Bersih

Teks bebas dari noise yang mengganggu, seperti header berulang, footer berulang, nomor halaman yang tidak perlu, watermark, karakter aneh, dan hasil OCR rusak.

2. Terstruktur

Dokumen masih memiliki heading, subheading, paragraf, daftar, tabel sederhana, dan urutan baca yang masuk akal.

3. Ber-metadata

Setiap dokumen dan chunk memiliki informasi sumber.

Contoh metadata minimal:

```
document_id: "DOC001"
title: "Panduan Akademik Mahasiswa"
source_file: "panduan-akademik.pdf"
page_start: 5
page_end: 6
section: "Pengisian KRS"
language: "id"
```

4. Terpotong rapi

Chunk tidak dipotong asal. Chunk harus cukup pendek untuk dicari, tetapi cukup panjang untuk mempertahankan konteks.

5. Bisa dilacak

Setiap jawaban harus dapat ditelusuri kembali ke sumber dokumen, halaman, atau bagian.

6. Memiliki knowledge layer

Untuk kebutuhan lebih lanjut, corpus dapat diperkaya dengan entitas, konsep, relasi, dan ontologi ringan.

Outline terlampir menekankan bahwa chunk RAG sebaiknya menyimpan `chunk_id`, `document_id`, `title`, `section`, `page_start`, `page_end`, dan `text`, agar setiap potongan teks tetap memiliki konteks dan sumber.

1.7 Mengapa Metadata Sangat Penting

Metadata adalah **identitas dan alamat** dari pengetahuan. Tanpa metadata, sistem RAG mungkin menemukan teks yang benar, tetapi sulit menjelaskan dari mana teks itu berasal.

Contoh chunk tanpa metadata:

```
Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.
```

Chunk ini berguna, tetapi tidak cukup kuat. Kita tidak tahu dari dokumen mana, halaman berapa, bagian apa, dan tahun berapa.

Contoh chunk dengan metadata:

```
{
  "chunk_id": "DOC001-0007",
  "document_id": "DOC001",
  "title": "Panduan Akademik Mahasiswa",
  "section": "Pengisian KRS",
  "page_start": 12,
  "page_end": 13,
  "text": "Mahasiswa wajib mengisi KRS sebelum perkuliahan
dimulai."
}
```

Versi kedua jauh lebih baik untuk RAG karena dapat digunakan untuk:

- pencarian,
- sitasi,
- audit jawaban,
- evaluasi kualitas,
- debugging,
- pembaruan corpus.

Metadata membuat jawaban LLM lebih dapat dipercaya.

1.8 Mengapa Ontologi Ringan Membantu Corpus RAG

Ontologi ringan tidak harus rumit. Untuk buku ini, ontologi cukup dipahami sebagai **peta konsep dan relasi**.

Contoh sederhana:

```
Mahasiswa → mengisi → KRS
Dosen Wali → menyetujui → KRS
KRS → berisi → Mata Kuliah
Mata Kuliah → memiliki → SKS
```

Relasi ini membantu mahasiswa melihat bahwa dokumen bukan hanya kumpulan paragraf, tetapi kumpulan pengetahuan. Dalam konteks RAG, ontologi ringan dapat membantu:

- membuat daftar istilah penting,
- menyusun sinonim,
- menghubungkan aktor dan proses,
- membuat pertanyaan uji,
- mengecek apakah jawaban LLM sesuai relasi sumber,
- memperkaya metadata corpus.

RDF dan OWL sering digunakan untuk merepresentasikan pengetahuan terstruktur. Referensi terlampir menjelaskan RDF sebagai bentuk triple **subject–predicate–object**, sedangkan OWL digunakan untuk menggambarkan kelas, karakteristik, dan hubungan antar-individu dalam Semantic Web. Theocharis dan Tsihrintzis juga menunjukkan penggunaan RDF, OWL, SPARQL, dan Protégé untuk akuisisi serta query pengetahuan berbasis ontologi.

Untuk mahasiswa, pesan utamanya adalah: **ontologi ringan bukan beban tambahan, tetapi alat bantu agar corpus lebih masuk akal.**

1.9 Pipeline Besar Buku

Pipeline utama buku ini adalah:

```
PDF
→ OCR
→ Markdown
→ Clean Corpus
→ Chunk
→ Metadata
→ Knowledge
→ Ontology Ringan
→ Upload RAG
```

Penjelasan ringkas:

Tahap	Tujuan
PDF	Sumber awal dokumen.
OCR	Membuat PDF scan menjadi searchable.
Markdown	Mengubah isi dokumen menjadi teks terstruktur.
Clean Corpus	Membersihkan teks dari noise.
Chunk	Memotong teks menjadi bagian yang mudah dicari.
Metadata	Memberi identitas, sumber, halaman, dan konteks.
Knowledge	Mengekstrak entitas, konsep, relasi, dan evidence.
Ontology Ringan	Membuat peta konsep dan relasi.
Upload RAG	Menyiapkan file final untuk Knowledge Base LLM.

Outline buku terlampir sudah mengarahkan workflow dari PDF ke OCR, Markdown, corpus, chunk, entitas, relasi, ontologi, dan upload RAG.

1.10 Studi Mini: PDF Panduan Akademik

Untuk Bab 1, mahasiswa belum perlu menulis banyak kode. Fokusnya adalah memahami masalah dan memilih bahan praktik.

Contoh dokumen yang cocok:

- panduan akademik,
- SOP laboratorium,
- brosur layanan kampus,
- aturan organisasi mahasiswa.

Contoh dokumen yang kurang cocok untuk pemula:

- PDF scan buram ratusan halaman,
- dokumen hukum sangat panjang,
- dokumen dengan banyak tabel kompleks,
- dokumen berisi banyak gambar tanpa caption,
- dokumen rahasia atau berisi data pribadi.

Saran awal: gunakan PDF 5–20 halaman, berbahasa Indonesia, dan topiknya familiar.

Praktik Mini Bab 1

Tujuan Praktik

Mahasiswa memilih satu PDF sederhana dan membuat dua file awal:

```
project_plan.md
sample_pdf_list.csv
```

Praktik ini belum masuk ke ekstraksi teknis. Tujuannya adalah membangun disiplin kerja: **mendokumentasikan sumber, tujuan, risiko, dan target output sebelum coding.**

File 1 — `project_plan.md`

Contoh isi:

```
# Project Plan Corpus RAG

## Judul Proyek
Konversi PDF Panduan Akademik menjadi Corpus RAG

## Tujuan
Membuat corpus bersih dari PDF agar dapat diupload ke sistem RAG LLM.

## Dokumen Sumber
- Nama file: panduan-akademik.pdf
- Jenis dokumen: panduan akademik
- Bahasa: Indonesia
- Perkiraan halaman: 20

## Masalah yang Mungkin Muncul
- Ada tabel
- Ada nomor halaman
- Ada header dan footer
- Struktur heading perlu diperiksa

## Output yang Ditargetkan
- clean_corpus.md
- chunks.jsonl
- metadata.csv
- entities.csv
- relations.csv
- corpus_for_rag.md

## Kriteria Berhasil
- Teks bersih dan mudah dibaca
- Chunk memiliki metadata
- Setiap jawaban RAG dapat dilacak ke sumber
```

File 2 — sample_pdf_list.csv

Contoh isi:

```
document_id,filename,document_type,language,estimated_pages,priority,notes
DOC001,panduan-akademik.pdf,panduan,id,20,high,Case utama untuk latihan
DOC002,sop-laboratorium.pdf,SOP,id,8,medium,Cocok untuk relasi aktor-proses
DOC003,brosur-layanan-kampus.pdf,brosur,id,4,low,Cocok untuk latihan ringan
```

Ringkasan Bab 1

Bab ini menegaskan bahwa **masalah utama RAG bukan hanya model LLM, tetapi kualitas corpus**. PDF mentah sering tidak cukup baik karena layout, scan, tabel, noise OCR, dan metadata yang hilang. Karena itu, PDF harus diubah menjadi corpus yang bersih, terstruktur, ber-metadata, terpotong rapi, dan memiliki jejak sumber.

Kalimat kunci Bab 1:

RAG yang kuat dimulai bukan dari model yang besar, tetapi dari corpus yang benar.

BAB 2 — Menyiapkan Workspace Python dan Tool Open Source

Workspace yang rapi adalah setengah dari keberhasilan corpus RAG; tanpa struktur folder, environment, dan tool yang jelas, pipeline PDF akan cepat menjadi kacau.

2.1 Dari Ide ke Lingkungan Kerja

Bab 1 sudah menegaskan bahwa **RAG yang kuat dimulai dari corpus yang benar**. Bab 2 masuk ke tahap berikutnya: menyiapkan lingkungan kerja agar mahasiswa dapat mengubah PDF menjadi corpus yang siap diproses.

Target Bab 2 sederhana: setelah selesai, mahasiswa harus punya:

- folder proyek yang rapi,
- *Python environment* yang terisolasi,
- tool OCR,
- tool ekstraksi PDF,
- tool konversi Markdown,
- tool dasar untuk metadata, chunking, dan knowledge extraction,
- catatan validasi bahwa semua tool berjalan.

Dokumen outline buku yang menjadi arah utama menempatkan Bab 2 sebagai bab praktis: membangun **workspace Python dan tool open source**, dengan output `environment_check.txt` dan `workspace_ready.md`.

2.2 Prinsip Kerja: Lokal, Terbuka, dan Bisa Diulang

Lingkungan kerja untuk corpus RAG harus mengikuti tiga prinsip.

1. Lokal

Sebisa mungkin proses dilakukan di komputer sendiri. Ini penting untuk:

- menjaga privasi dokumen,
- mengurangi ketergantungan layanan luar,
- memudahkan eksperimen,
- mempercepat debugging.

2. Open Source First

Tool yang dipakai harus mudah dipelajari, dapat diaudit, dan tidak mengunci mahasiswa pada satu ekosistem tertutup.

Tool utama dalam bab ini:

- **Python**
- **OCRmyPDF**
- **Tesseract**
- **PyMuPDF**
- **pdfplumber**
- **pandas**
- **spaCy**
- **rdflib**
- **Docling**
- **Open WebUI**

OCRmyPDF digunakan karena dokumentasi resminya menyatakan bahwa tool ini menambahkan lapisan teks OCR ke PDF scan agar dapat dicari. ([OCRmyPDF](#))

Docling digunakan karena mendukung parsing berbagai format dokumen, termasuk PDF, memahami layout, reading order, tabel, formula, serta dapat mengekspor ke Markdown dan JSON. ([Docling Project](#))

3. Bisa Diulang

Semua perintah, folder, dan output harus bisa dibuat ulang. Jangan mengandalkan langkah manual yang tidak tercatat.

Catatan kritis: pipeline yang tidak bisa diulang tidak layak dipakai untuk riset, skripsi, atau proyek AI serius.

2.3 Struktur Folder Proyek

Struktur folder adalah fondasi. Jangan mencampur PDF mentah, hasil OCR, teks bersih, chunk, dan file upload RAG dalam satu folder.

Gunakan struktur berikut:

```
pdf-to-rag-corpus/  
├── data/  
│   ├── pdf/  
│   ├── ocr/  
│   ├── raw_text/  
│   ├── markdown/  
│   ├── clean/  
│   ├── chunks/  
│   ├── metadata/  
│   ├── knowledge/  
│   ├── ontology/  
│   └── rag_upload/  
├── scripts/  
├── notebooks/  
├── output/  
└── README.md
```

Fungsi Setiap Folder

Folder	Fungsi
<code>data/pdf/</code>	Menyimpan PDF asli. Jangan diedit.
<code>data/ocr/</code>	Menyimpan PDF hasil OCR.
<code>data/raw_text/</code>	Menyimpan teks mentah hasil ekstraksi.
<code>data/markdown/</code>	Menyimpan hasil konversi ke Markdown.
<code>data/clean/</code>	Menyimpan corpus yang sudah dibersihkan.
<code>data/chunks/</code>	Menyimpan file chunk untuk RAG.
<code>data/metadata/</code>	Menyimpan index, laporan kualitas, dan metadata.
<code>data/knowledge/</code>	Menyimpan entitas, konsep, relasi, dan evidence.
<code>data/ontology/</code>	Menyimpan RDF, Turtle, OWL, atau diagram relasi.
<code>data/rag_upload/</code>	Menyimpan file final yang siap upload ke RAG.
<code>scripts/</code>	Menyimpan script Python.
<code>notebooks/</code>	Menyimpan notebook eksperimen.

<code>output/</code>	Menyimpan laporan akhir.
<code>README.md</code>	Menjelaskan isi proyek.

Prinsip penting: folder `data/pdf/` adalah **sumber asli**. Jangan menimpa file asli. Semua hasil kerja harus masuk ke folder turunannya.

2.4 Membuat Folder Proyek

Jalankan perintah berikut di terminal Linux.

```
mkdir -p
pdf-to-rag-corpus/data/{pdf,ocr,raw_text,markdown,clean,chunks,metadata,knowledge,ontology,rag_upload}
mkdir -p pdf-to-rag-corpus/{scripts,notebooks,output}
touch pdf-to-rag-corpus/README.md
cd pdf-to-rag-corpus
```

Penjelasan Perintah

- `mkdir` berarti membuat folder.
- `-p` berarti membuat folder bertingkat tanpa error jika folder sudah ada.
- `{pdf,ocr,...}` adalah pola ringkas untuk membuat banyak folder sekaligus.
- `touch README.md` membuat file kosong bernama `README.md`.
- `cd pdf-to-rag-corpus` masuk ke folder proyek.

Output yang diharapkan:

```
pdf-to-rag-corpus/
├── data/
├── scripts/
├── notebooks/
├── output/
└── README.md
```

2.5 Instalasi Python dan Virtual Environment

Python dipakai sebagai bahasa utama karena ekosistemnya kuat untuk ekstraksi dokumen, pembersihan teks, metadata, NLP, dan RDF.

Gunakan *virtual environment* agar paket proyek ini tidak bercampur dengan proyek lain. Dokumentasi resmi Python menjelaskan bahwa `venv` membuat lingkungan Python ringan dan terisolasi, dengan paket sendiri yang terpisah dari instalasi Python dasar. ([Python documentation](#))

Membuat Virtual Environment

```
python3 -m venv .venv
source .venv/bin/activate
python --version
pip --version
```

Penjelasan Perintah

- `python3 -m venv .venv` membuat folder environment bernama `.venv`.
- `source .venv/bin/activate` mengaktifkan environment.
- `python --version` mengecek versi Python.
- `pip --version` mengecek pengelola paket Python.

Jika berhasil, prompt terminal biasanya menampilkan tanda:

```
(.venv)
```

Catatan praktik: nama `.venv` disarankan karena umum dipakai, rapi, dan mudah dikenali sebagai folder environment proyek. Dokumentasi tutorial Python juga menyebut `.venv` sebagai lokasi umum untuk virtual environment. ([Python documentation](#))

2.6 Instalasi OCRmyPDF dan Tesseract

PDF scan tidak memiliki teks. Agar dapat dicari, PDF scan perlu diberi lapisan teks OCR.

OCRmyPDF bekerja dengan Tesseract OCR. Dokumentasi OCRmyPDF menyatakan bahwa OCRmyPDF menggunakan Tesseract sebagai OCR engine untuk melakukan OCR pada PDF. ([OCRmyPDF](#))

Instalasi di Ubuntu / Debian

```
sudo apt update
sudo apt install -y ocrmypdf tesseract-ocr tesseract-ocr-ind
tesseract-ocr-eng
```

Penjelasan Perintah

- `sudo apt update` memperbarui daftar paket.
- `sudo apt install -y` memasang paket tanpa banyak konfirmasi.
- `ocrmypdf` adalah tool OCR untuk PDF.
- `tesseract-ocr` adalah mesin OCR.
- `tesseract-ocr-ind` adalah data bahasa Indonesia.
- `tesseract-ocr-eng` adalah data bahasa Inggris.

Tes OCRmyPDF

```
ocrmypdf --version
tesseract --version
tesseract --list-langs
```

Output penting yang harus terlihat:

```
ind
eng
```

Jika `ind` tidak muncul, OCR bahasa Indonesia belum terpasang.

2.7 Instalasi Library Python Utama

Setelah environment aktif, pasang library Python.

```
pip install --upgrade pip
pip install pymupdf pdfplumber pandas spacy rdflib
```

Fungsi Library

Library	Fungsi
pymupdf	Membaca PDF dan mengekstrak teks per halaman.
pdfplumber	Mengekstrak teks dan tabel dari PDF.
pandas	Mengolah CSV, metadata, dan tabel.
spacy	NLP dasar: token, kalimat, entitas, pola bahasa.
rdflib	Membuat RDF/Turtle untuk ontologi ringan.

Instal Model Bahasa spaCy

Untuk awal, gunakan model kecil bahasa Inggris jika diperlukan untuk eksperimen. Untuk bahasa Indonesia, mahasiswa dapat memakai pendekatan rule-based sederhana atau tool NLP lain yang mendukung bahasa Indonesia.

```
python -m spacy download en_core_web_sm
```

Catatan kritis: jangan memaksa semua ekstraksi entitas bahasa Indonesia memakai model yang tidak cocok. Untuk corpus Indonesia, pendekatan awal yang aman adalah gabungan **heading extraction**, **keyword frequency**, **pola kalimat**, dan **validasi manual**.

2.8 Instalasi dan Penggunaan Docling

Docling dipakai untuk mengubah PDF menjadi representasi yang lebih siap untuk AI, terutama Markdown dan JSON. Dokumentasi Docling menyebut dukungan parsing PDF, layout, reading order, tabel, code, formula, gambar, serta ekspor ke Markdown dan JSON. ([Docling Project](#))

Instalasi Docling

```
pip install docling
```

Tes Docling

```
docling --version
```

Contoh Konversi PDF ke Markdown

Masukkan satu PDF contoh ke:

```
data/pdf/sample.pdf
```

Lalu jalankan:

```
docling data/pdf/sample.pdf --to md --output  
data/markdown/
```

Output yang diharapkan:

```
data/markdown/sample.md
```

Catatan praktik: jika PDF memiliki layout kompleks, Docling sering lebih berguna daripada ekstraksi teks biasa, karena struktur dokumen lebih terjaga.

2.9 Pengenalan Open WebUI Knowledge Base

Open WebUI digunakan sebagai tempat menguji corpus RAG. Dokumentasi resminya menjelaskan bahwa *Knowledge Base* dibuat melalui menu Workspace, lalu pengguna dapat upload file atau menambahkan dokumen, dan knowledge dapat direferensikan di chat dengan #. ([Open WebUI](#))

Untuk buku ini, Open WebUI dipakai pada tahap akhir pipeline:

```
clean corpus
→ chunks
→ metadata
→ rag_upload/
→ upload ke Knowledge Base
→ uji tanya-jawab
```

Open WebUI juga mendokumentasikan bahwa file dapat digunakan untuk RAG, baik sebagai upload dalam chat maupun sebagai *Knowledge Base* yang lebih terorganisasi. ([Open WebUI](#))

File yang Akan Disiapkan untuk Upload

Pada bab-bab berikutnya, mahasiswa akan membuat:

```
data/rag_upload/
├─ corpus_for_rag.md
├─ corpus_all.jsonl
├─ metadata.csv
└─ upload_readme.md
```

Prinsip penting: jangan upload semua file kerja. Upload hanya file final yang sudah bersih dan siap dicari.

2.10 Membuat File Requirements

Agar environment mudah dibuat ulang, simpan daftar paket Python.

```
pip freeze > requirements.txt
```

Isi file ini akan mirip seperti:

```
pymupdf==...  
pdfplumber==...  
pandas==...  
spacy==...  
rdflib==...  
docling==...
```

Untuk memasang ulang di komputer lain:

```
pip install -r requirements.txt
```

Penjelasan Perintah

- `pip freeze` menampilkan daftar paket yang terpasang.
- `>` menyimpan output ke file.
- `requirements.txt` adalah daftar dependensi proyek.
- `pip install -r requirements.txt` memasang semua paket dari file.

Catatan strategis: proyek RAG yang baik harus bisa dipindahkan dan direplikasi.

2.11 Script Cek Environment

Buat file:

scripts/check_environment.py

Isi script:

```
import importlib.util
import shutil
from pathlib import Path

TOOLS = ["python3", "ocrmypdf", "tesseract", "docling"]

PYTHON_PACKAGES = [
    "fitz",          # PyMuPDF
    "pdfplumber",
    "pandas",
    "spacy",
    "rdflib",
]

PROJECT_DIRS = [
    "data/pdf",
    "data/ocr",
    "data/raw_text",
    "data/markdown",
    "data/clean",
    "data/chunks",
    "data/metadata",
    "data/knowledge",
    "data/ontology",
    "data/rag_upload",
    "scripts",
    "notebooks",
    "output",
]

def check_command(command):
    return shutil.which(command) is not None

def check_package(package):
    return importlib.util.find_spec(package) is not None

def main():
    lines = []
    lines.append("# Environment Check\n")

    lines.append("## Command Line Tools")
    for tool in TOOLS:
        status = "OK" if check_command(tool) else "MISSING"
        lines.append(f"- {tool}: {status}")

    lines.append("\n## Python Packages")
    for package in PYTHON_PACKAGES:
        status = "OK" if check_package(package) else "MISSING"
        lines.append(f"- {package}: {status}")
```

```

lines.append("\n## Project Folders")
for folder in PROJECT_DIRS:
    status = "OK" if Path(folder).exists() else "MISSING"
    lines.append(f"- {folder}: {status}")

report = "\n".join(lines)
Path("output").mkdir(exist_ok=True)
Path("output/environment_check.txt").write_text(report,
encoding="utf-8")
print(report)

if __name__ == "__main__":
    main()

```

Jalankan:

```
python scripts/check_environment.py
```

Output:

```
output/environment_check.txt
```

Penjelasan Ringkas Script

- `importlib.util` mengecek apakah paket Python terpasang.
- `shutil.which()` mengecek apakah perintah terminal tersedia.
- `Path()` dipakai untuk mengecek folder.
- `TOOLS` berisi tool command line.
- `PYTHON_PACKAGES` berisi paket Python.
- `PROJECT_DIRS` berisi folder wajib.
- `write_text()` menyimpan laporan ke file.

Tujuan script ini bukan keren-kerenan. Tujuannya disiplin: sebelum memproses PDF, pastikan alat kerja sudah benar.

2.12 Membuat File workspace_ready.md

Buat file:

```
output/workspace_ready.md
```

Contoh isi:

```
# Workspace Ready Report

## Nama Proyek
pdf-to-rag-corpus

## Tujuan
Menyiapkan workspace untuk konversi PDF menjadi corpus siap upload ke RAG
LLM.

## Status Folder
- data/pdf: siap
- data/ocr: siap
- data/raw_text: siap
- data/markdown: siap
- data/clean: siap
- data/chunks: siap
- data/metadata: siap
- data/knowledge: siap
- data/ontology: siap
- data/rag_upload: siap

## Tool Utama
- Python: siap
- OCRmyPDF: siap
- Tesseract: siap
- Docling: siap
- PyMuPDF: siap
- pdfplumber: siap
- pandas: siap
- spaCy: siap
- rdflib: siap

## Catatan
Workspace siap digunakan untuk Bab 3: Audit Kualitas PDF.
```

File ini menjadi bukti bahwa proyek siap masuk tahap berikutnya.

2.13 Praktik Utama Bab 2

Tugas Praktik

Mahasiswa harus melakukan empat hal:

1. Membuat struktur folder proyek.
2. Mengaktifkan *virtual environment* Python.
3. Menginstal tool OCR dan library Python.
4. Menjalankan script cek environment.

Output Wajib

output/environment_check.txt
output/workspace_ready.md
requirements.txt

Kriteria Berhasil

Bab ini dianggap berhasil jika:

- semua folder utama tersedia,
- Python environment aktif,
- OCRmyPDF berjalan,
- Tesseract berjalan,
- bahasa OCR minimal **ind** dan **eng** tersedia,
- library Python utama dapat diimpor,
- Docling dapat dipanggil,
- laporan environment berhasil dibuat.

2.14 Troubleshooting Ringkas

Masalah	Penyebab Umum	Solusi
<code>python3: command not found</code>	Python belum terpasang	Instal Python dari repositori distro Linux.
<code>pip: command not found</code>	pip belum tersedia	Pasang paket pip sesuai distro.
<code>ocrmypdf: command not found</code>	OCRmyPDF belum terpasang	Instal OCRmyPDF.
<code>tesseract: command not found</code>	Tesseract belum terpasang	Instal Tesseract OCR.
Bahasa <code>ind</code> tidak muncul	Data bahasa Indonesia belum terpasang	Instal paket bahasa Indonesia Tesseract.
<code>ModuleNotFoundError</code>	Paket Python belum terpasang di environment aktif	Aktifkan <code>.venv</code> , lalu jalankan <code>pip install</code> .
Docling gagal membaca PDF	PDF rusak, terenkripsi, atau layout terlalu kompleks	Coba PDF lain atau audit di Bab 3.

Prinsip debugging: jangan langsung menyalahkan LLM. Cek dulu folder, environment, file, dan tool.

Ringkasan Bab 2

Bab ini membangun fondasi teknis buku. Mahasiswa menyiapkan workspace, *virtual environment*, tool OCR, library PDF, library NLP, library RDF, dan Docling. Semua diarahkan untuk satu tujuan: membuat pipeline yang bisa diulang dari PDF mentah menuju corpus siap upload ke RAG.

Kalimat kunci Bab 2:

AI yang baik membutuhkan data yang baik; data yang baik membutuhkan workspace yang rapi.

BAB 3 — Audit Kualitas PDF sebelum Diproses

Jangan mulai ekstraksi sebelum tahu kualitas PDF; RAG yang gagal sering dimulai dari PDF yang tidak diaudit.

3.1 Audit Dulu, Proses Kemudian

Banyak mahasiswa langsung masuk ke tahap ekstraksi PDF. Ini keliru. Sebelum PDF diubah menjadi *corpus*, kita harus tahu dulu apakah PDF tersebut berisi teks asli, hasil scan, tabel kompleks, gambar, dua kolom, atau halaman buram.

Dalam arah buku ini, Bab 3 memang ditempatkan sebagai tahap **audit kualitas PDF** sebelum masuk ke Bab 4 tentang ekstraksi teks dan Markdown. Output utamanya adalah `metadata/pdf_quality_report.csv`, dengan kolom `filename`, `pages`, `has_text`, `avg_text_length`, `suspected_scan`, `needs_ocr`, dan `notes`.

Prinsipnya sederhana:

LLM tidak bisa mencari informasi yang sejak awal gagal di ekstrak.

3.2 PDF Searchable vs PDF Scan

Ada dua jenis PDF yang paling sering ditemukan.

1. PDF Searchable

PDF *searchable* adalah PDF yang memiliki lapisan teks. Teks di dalamnya dapat disalin, dicari, dan diekstrak oleh Python.

Ciri umum:

- teks bisa di-*select*,
- kata bisa dicari,
- ukuran file relatif efisien,
- hasil ekstraksi biasanya lebih baik,
- cocok diproses dengan PyMuPDF atau pdfplumber.

PyMuPDF menyediakan fungsi ekstraksi teks dari PDF, misalnya dengan membuka dokumen lalu mengambil teks per halaman memakai `page.get_text()`.

([PyMuPDF Documentation](#))

2. PDF Scan

PDF scan biasanya hanya berisi gambar halaman. Secara visual terlihat seperti teks, tetapi bagi komputer ia hanyalah gambar.

Ciri umum:

- teks tidak bisa di-*select*,
- pencarian kata tidak bekerja,
- hasil ekstraksi kosong atau sangat pendek,
- perlu OCR,
- rawan salah baca jika scan buram.

OCRmyPDF relevan untuk kasus ini karena tool tersebut menambahkan lapisan teks OCR ke PDF scan. Dokumentasi OCRmyPDF juga menjelaskan opsi `--skip-text` untuk melewati halaman yang sudah memiliki teks dan `--force-ocr` untuk memaksa OCR pada halaman tertentu. ([OCRmyPDF](#))

3.3 PDF Dua Kolom

PDF dua kolom sering muncul pada jurnal, paper, laporan teknis, dan prosiding. Masalahnya bukan hanya teks ada atau tidak, tetapi **urutan baca**.

Contoh masalah:

Kolom kiri paragraf 1 + kolom kanan paragraf 1 + kolom kiri paragraf 2

Padahal urutan yang benar adalah:

Kolom kiri selesai dulu, baru kolom kanan.

Dampaknya untuk RAG:

- chunk menjadi kacau,
- kalimat tidak nyambung,
- konteks hilang,
- LLM menjawab berdasarkan potongan yang salah urut.

Catatan kritis: PDF dua kolom tidak selalu gagal, tetapi wajib ditandai sejak audit.

3.4 PDF dengan Tabel

Tabel adalah sumber masalah besar untuk corpus RAG. Manusia bisa membaca baris dan kolom secara visual. Namun, hasil ekstraksi sering berubah menjadi teks linear yang kehilangan struktur.

Contoh tabel asli:

Layanan	Syarat	Waktu
KRS	Login akun akademik	3 hari
KHS	Status aktif	1 hari

Hasil ekstraksi buruk bisa menjadi:

```
Layanan Syarat Waktu KRS Login akun akademik 3 hari KHS
Status aktif 1 hari
```

pdfplumber berguna untuk PDF berbasis teks karena dokumentasinya menyebut kemampuan mengambil informasi detail seperti karakter, garis, rectangle, serta ekstraksi tabel dan *visual debugging*. ([GitHub](#))

Untuk audit awal, mahasiswa belum perlu mengekstrak tabel sempurna. Cukup tandai:

- ada tabel,
- tabel banyak,
- tabel penting,
- tabel kemungkinan perlu diproses khusus.

Tabel penting tidak boleh dibiarkan menjadi paragraf acak.

3.5 PDF dengan Gambar dan Diagram

Gambar dan diagram sering membawa informasi penting, tetapi tidak otomatis berubah menjadi teks.

Contoh:

- alur proses akademik,
- diagram organisasi,
- flowchart SOP,
- skema jaringan,
- grafik hasil survei.

Masalahnya:

- ekstraksi teks tidak membaca gambar,
- OCR belum tentu memahami diagram,
- caption bisa terpisah dari gambar,
- informasi visual hilang dari corpus.

Untuk tahap awal, audit cukup mencatat:

```
contains_images = yes
notes = "Ada diagram proses pada beberapa halaman; perlu
caption manual."
```

Strategi praktis: jika gambar penting, buat ringkasan manual atau caption terstruktur pada Bab 5.

3.6 PDF dengan Halaman Miring, Buram, atau Pecah

PDF scan yang miring, buram, atau pecah akan menghasilkan OCR buruk.

Tanda-tandanya:

- teks tampak miring,
- huruf pecah,
- kontras rendah,
- bayangan kertas terlihat,
- sebagian halaman terpotong,
- hasil OCR banyak karakter aneh.

Dampaknya:

- kata penting berubah,
- angka salah,
- nama orang salah,
- istilah teknis rusak,
- metadata halaman tidak akurat.

Tesseract mendukung banyak bahasa melalui file `traineddata`, termasuk `ind` untuk bahasa Indonesia. Ini penting agar OCR dokumen Indonesia tidak dipaksa memakai model bahasa yang tidak sesuai. ([GitHub](#))

Pesan tajam: OCR bukan sihir. Scan buruk tetap menghasilkan corpus buruk.

3.7 Mengecek Jumlah Halaman dan Panjang Teks per Halaman

Audit PDF minimal harus menjawab:

- Berapa jumlah halaman?
- Apakah setiap halaman memiliki teks?
- Berapa rata-rata panjang teks per halaman?
- Apakah dokumen diduga scan?
- Apakah perlu OCR?
- Apakah ada catatan khusus?

Indikasi sederhana:

Kondisi	Interpretasi
Rata-rata teks per halaman tinggi	Kemungkinan <i>searchable</i>
Rata-rata teks per halaman sangat rendah	Kemungkinan scan
Beberapa halaman kosong	Perlu cek manual
Banyak halaman pendek	Bisa berisi tabel, gambar, atau scan
Teks ada tetapi kacau	Perlu ekstraksi alternatif

Contoh ambang awal:

```
avg_text_length < 50 karakter per halaman → suspected_scan =  
yes
```

Ambang ini bukan hukum mutlak. Ini hanya indikator awal untuk audit.

3.8 Menentukan Apakah PDF Perlu OCR

Gunakan logika sederhana:

Jika PDF tidak punya teks → perlu OCR.
Jika teks sangat sedikit → kemungkinan perlu OCR.
Jika teks ada tetapi rusak → perlu cek manual.
Jika PDF sudah searchable → tidak perlu OCR penuh.

Rekomendasi:

- PDF scan: gunakan OCR.
- PDF campuran teks + scan: gunakan OCR dengan hati-hati.
- PDF sudah searchable: langsung ekstraksi.
- PDF hasil OCR buruk: pertimbangkan `--force-ocr` pada salinan file, bukan file asli.

OCRmyPDF menyediakan opsi `--skip-text` untuk melewati halaman yang sudah memiliki teks dan `--force-ocr` untuk memaksa OCR ketika diperlukan, misalnya jika hasil OCR sebelumnya buruk atau ada watermark teks. ([OCRmyPDF](#))

Jangan menimpa PDF asli. Simpan hasil OCR di folder `data/ocr/`.

3.9 Praktik Python Bab 3

Tujuan Praktik

Mahasiswa membuat script untuk membaca semua PDF dalam folder:

```
data/pdf/
```

Lalu menghasilkan laporan:

```
data/metadata/pdf_quality_report.csv
```

Kolom laporan:

```
filename, pages, has_text, avg_text_length,  
suspected_scan, needs_ocr, notes
```

Script: `scripts/audit_pdf_quality.py`

```
from pathlib import Path  
import fitz  
import pandas as pd  
  
PDF_DIR = Path("data/pdf")  
OUTPUT_DIR = Path("data/metadata")  
OUTPUT_FILE = OUTPUT_DIR / "pdf_quality_report.csv"  
  
MIN_AVG_TEXT_LENGTH = 50  
  
def extract_page_text_lengths(pdf_path):  
    doc = fitz.open(pdf_path)  
    page_lengths = []  
  
    for page in doc:  
        text = page.get_text("text")  
        page_lengths.append(len(text.strip()))  
  
    doc.close()  
    return page_lengths  
  
def audit_pdf(pdf_path):  
    try:  
        page_lengths = extract_page_text_lengths(pdf_path)  
        pages = len(page_lengths)  
  
        total_text_length = sum(page_lengths)  
        avg_text_length = total_text_length / pages if pages > 0 else 0  
  
        has_text = total_text_length > 0  
        suspected_scan = avg_text_length < MIN_AVG_TEXT_LENGTH  
        needs_ocr = suspected_scan
```

```

        notes = []

        if pages == 0:
            notes.append("PDF tidak memiliki halaman terbaca")

        if not has_text:
            notes.append("Tidak ada teks terdeteksi")

        if suspected_scan:
            notes.append("Kemungkinan PDF scan atau image-only")

        if avg_text_length < MIN_AVG_TEXT_LENGTH and has_text:
            notes.append("Teks sangat sedikit; perlu cek manual")

        return {
            "filename": pdf_path.name,
            "pages": pages,
            "has_text": has_text,
            "avg_text_length": round(avg_text_length, 2),
            "suspected_scan": suspected_scan,
            "needs_ocr": needs_ocr,
            "notes": "; ".join(notes) if notes else "OK",
        }

    except Exception as error:
        return {
            "filename": pdf_path.name,
            "pages": 0,
            "has_text": False,
            "avg_text_length": 0,
            "suspected_scan": True,
            "needs_ocr": True,
            "notes": f"ERROR: {error}",
        }

def main():
    OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

    pdf_files = sorted(PDF_DIR.glob("*.pdf"))

    results = []
    for pdf_file in pdf_files:
        results.append(audit_pdf(pdf_file))

    df = pd.DataFrame(results)
    df.to_csv(OUTPUT_FILE, index=False, encoding="utf-8")

    print(f"Laporan berhasil dibuat: {OUTPUT_FILE}")

if __name__ == "__main__":
    main()

```

3.10 Penjelasan Script

Bagian 1 — Import Library

```
from pathlib import Path
import fitz
import pandas as pd
```

- **Path** dipakai untuk mengelola lokasi file dan folder.
- **fitz** adalah modul PyMuPDF untuk membuka dan membaca PDF.
- **pandas** dipakai untuk membuat tabel dan menyimpan CSV.

Bagian 2 — Lokasi Folder

```
PDF_DIR = Path("data/pdf")
OUTPUT_DIR = Path("data/metadata")
OUTPUT_FILE = OUTPUT_DIR / "pdf_quality_report.csv"
```

- **PDF_DIR** adalah folder PDF sumber.
- **OUTPUT_DIR** adalah folder output metadata.
- **OUTPUT_FILE** adalah nama laporan akhir.

Bagian 3 — Ambang Deteksi Scan

```
MIN_AVG_TEXT_LENGTH = 50
```

Angka ini berarti: jika rata-rata teks per halaman kurang dari 50 karakter, PDF dicurigai sebagai scan atau dokumen yang miskin teks.

Catatan: angka ini boleh disesuaikan setelah melihat jenis dokumen.

Bagian 4 — Ekstraksi Panjang Teks per Halaman

```
def extract_page_text_lengths(pdf_path):
    doc = fitz.open(pdf_path)
    page_lengths = []

    for page in doc:
        text = page.get_text("text")
        page_lengths.append(len(text.strip()))

    doc.close()
    return page_lengths
```

Fungsi ini:

- membuka PDF,
- membaca setiap halaman,
- mengambil teks,
- menghitung panjang teks,
- mengembalikan daftar panjang teks per halaman.

Bagian 5 — Audit Satu PDF

```
def audit_pdf(pdf_path):
```

Fungsi ini mengecek satu file PDF dan menghasilkan satu baris laporan.

Logika penting:

```
has_text = total_text_length > 0
suspected_scan = avg_text_length < MIN_AVG_TEXT_LENGTH
needs_ocr = suspected_scan
```

Artinya:

- jika total teks lebih dari nol, PDF dianggap punya teks,
- jika rata-rata teks sangat rendah, PDF dicurigai scan,
- jika dicurigai scan, PDF perlu OCR.

Bagian 6 — Menyimpan CSV

```
df = pd.DataFrame(results)
df.to_csv(OUTPUT_FILE, index=False, encoding="utf-8")
```

Bagian ini mengubah hasil audit menjadi tabel dan menyimpannya sebagai CSV.

3.11 Menjalankan Script

Pastikan file PDF sudah ada di:

```
data/pdf/
```

Jalankan:

```
python scripts/audit_pdf_quality.py
```

Output:

```
data/metadata/pdf_quality_report.csv
```

Contoh isi:

```
filename,pages,has_text,avg_text_length,suspected_scan,needs_ocr,notes
panduan-akademik.pdf,20,True,1240.55,False,False,OK
sop-lab-scan.pdf,8,False,0.0,True,True,Tidak ada teks terdeteksi; Kemungkinan PDF scan atau image-only
brosur-layanan.pdf,4,True,35.25,True,True,Teks sangat sedikit; perlu cek manual
```

3.12 Cara Membaca Hasil Audit

Kasus 1 — PDF Baik

```
has_text = True
avg_text_length = 1240.55
suspected_scan = False
needs_ocr = False
notes = OK
```

Artinya PDF bisa langsung masuk ke Bab 4 untuk ekstraksi teks.

Kasus 2 — PDF Scan

```
has_text = False
avg_text_length = 0
suspected_scan = True
needs_ocr = True
```

Artinya PDF harus diproses OCR lebih dulu.

Kasus 3 — PDF Meragukan

```
has_text = True
avg_text_length = 35.25
suspected_scan = True
needs_ocr = True
notes = Teks sangat sedikit; perlu cek manual
```

Artinya PDF mungkin berisi gambar, tabel, atau hasil OCR buruk. Perlu dilihat manual.

3.13 Checklist Audit Manual

Script membantu audit awal, tetapi tidak menggantikan mata manusia. Mahasiswa tetap perlu membuka beberapa halaman contoh.

Checklist manual:

- Apakah teks bisa di-*select*?
- Apakah urutan baca benar?
- Apakah ada dua kolom?
- Apakah tabel penting masih terbaca?
- Apakah ada gambar atau diagram penting?
- Apakah halaman miring atau buram?
- Apakah ada header-footer berulang?
- Apakah dokumen berisi data sensitif?

Audit manual cukup 3–5 halaman contoh: awal, tengah, akhir, dan halaman yang terlihat kompleks.

3.14 Status Keputusan Setelah Audit

Setelah audit, beri keputusan sederhana:

Status	Arti	Tindakan
READY_EXTRACT	PDF punya teks cukup baik	Masuk Bab 4
NEEDS_OCR	PDF scan atau teks sangat sedikit	OCR dulu
NEEDS_MANUAL_CHECK	PDF kompleks atau meragukan	Cek manual
REJECTED	PDF rusak, terlalu buram, atau tidak layak	Ganti sumber

Kolom ini bisa ditambahkan nanti di versi lanjutan:

```
decision
```

Namun untuk Bab 3, kolom wajib tetap mengikuti outline:

```
filename, pages, has_text, avg_text_length, suspected_scan,  
needs_ocr, notes
```

Output Bab 3

Output wajib:

```
data/metadata/pdf_quality_report.csv
```

Output tambahan yang disarankan:

```
output/pdf_audit_notes.md
```

Contoh isi `pdf_audit_notes.md`:

```
# Catatan Audit PDF

## DOC001 - panduan-akademik.pdf
Status: READY_EXTRACT

Catatan:
- Teks terbaca.
- Ada beberapa tabel.
- Ada header dan footer.
- Cocok untuk latihan Bab 4.

## DOC002 - sop-lab-scan.pdf
Status: NEEDS_OCR

Catatan:
- Tidak ada teks terdeteksi.
- Kemungkinan hasil scan.
- Perlu OCR sebelum ekstraksi.
```

Ringkasan Bab 3

Bab ini mengajarkan mahasiswa untuk tidak gegabah memproses PDF. PDF harus diaudit lebih dulu: apakah *searchable*, scan, dua kolom, banyak tabel, banyak gambar, atau buram. Hasil audit disimpan dalam `pdf_quality_report.csv` agar keputusan teknis di Bab 4 tidak berdasarkan tebakan.

Kalimat kunci Bab 3:

Sebelum corpus dibersihkan, PDF harus diperiksa; sebelum RAG dipercaya, sumbernya harus diaudit.

BAB 4 — Ekstraksi PDF ke Teks dan Markdown

Tidak ada satu tool yang sempurna untuk semua PDF; corpus RAG yang baik lahir dari proses ekstraksi yang diuji, dibandingkan, dan dipilih secara sadar.

4.1 Bab Inti Teknis Pertama

Bab ini adalah titik masuk utama ke pekerjaan teknis. Setelah Bab 3 mahasiswa melakukan audit kualitas PDF, Bab 4 mengajarkan cara mengekstrak isi PDF menjadi **teks mentah** dan **Markdown**.

Arah buku sudah jelas: Bab 4 harus membandingkan beberapa jalur ekstraksi, yaitu **PyMuPDF**, **pdfplumber**, **OCRmyPDF**, dan **Docling**, lalu menghasilkan `raw_text/document_001.txt`, `markdown/document_001.md`, dan `metadata/extraction_comparison.csv`.

Pesan utama bab ini: **jangan fanatik pada satu tool**. PDF *searchable*, PDF scan, PDF tabel, dan PDF kompleks membutuhkan pendekatan berbeda.

4.2 Mengapa Perlu Banyak Pipeline

PDF bukan satu jenis dokumen. Ada PDF yang bersih, ada yang hasil scan, ada yang dua kolom, ada yang penuh tabel, ada yang memiliki diagram, dan ada yang urutan bacanya rusak.

Karena itu, bab ini memakai tiga pipeline utama:

```
Pipeline A: PDF searchable → PyMuPDF → raw_text.txt
Pipeline B: PDF scan → OCRmyPDF → searchable.pdf →
raw_text.txt
Pipeline C: PDF kompleks → Docling → markdown.md
```

PyMuPDF cocok untuk ekstraksi cepat dari PDF yang sudah memiliki teks. Dokumentasi PyMuPDF menjelaskan bahwa `Page.get_text()` dapat mengekstrak teks, blok, kata, dan informasi posisi; tetapi urutan teks mentah belum tentu sama dengan urutan baca alami, sehingga opsi `sort=True` dapat dipertimbangkan. ([PyMuPDF Documentation](#))

pdfplumber cocok ketika mahasiswa perlu membaca teks, karakter, garis, rectangle, dan tabel dari PDF berbasis teks; dokumentasinya juga menekankan fitur ekstraksi tabel serta *visual debugging*. ([GitHub](#))

OCRmyPDF cocok untuk PDF scan karena menambahkan lapisan teks OCR agar PDF dapat dicari. ([OCRmyPDF](#))

Docling cocok untuk workflow modern AI/RAG karena dokumentasinya menyebut parsing berbagai format, pemahaman PDF tingkat lanjut, *reading order*, struktur tabel, formula, serta ekspor ke Markdown dan JSON. ([Docling Project](#))

4.3 Prinsip Ekstraksi untuk Corpus RAG

Ekstraksi PDF untuk RAG berbeda dari sekadar menyalin teks.

Tujuan kita bukan hanya mendapatkan teks, tetapi mendapatkan teks yang:

- mudah dibaca,
- urutannya masuk akal,
- heading tetap terlihat,
- tabel tidak hancur total,
- bisa dibersihkan di Bab 5,
- bisa di-*chunk* di Bab 6,
- bisa dilacak ke sumber.

Dokumen awal buku juga menekankan bahwa output utama proses PDF adalah **Markdown**, karena mudah dibaca, mudah diberi metadata, dan cocok untuk versioning serta konversi lanjutan.

Kesalahan umum: mahasiswa terlalu cepat percaya hasil ekstraksi pertama. Padahal, ekstraksi pertama sering hanya bahan mentah.

4.4 Persiapan Folder

Pastikan struktur folder dari Bab 2 sudah ada:

```
pdf-to-rag-corpus/  
├── data/  
│   ├── pdf/  
│   ├── ocr/  
│   ├── raw_text/  
│   ├── markdown/  
│   └── metadata/  
├── scripts/  
└── output/
```

Masukkan satu file contoh ke:

```
data/pdf/document_001.pdf
```

Untuk latihan awal, gunakan PDF sederhana:

- panduan akademik,
- SOP laboratorium,
- brosur layanan kampus,
- aturan organisasi mahasiswa.

Jangan mulai dari PDF ratusan halaman. Mulai dari 5–20 halaman agar mudah diperiksa.

4.5 Pipeline A — Ekstraksi PDF Searchable dengan PyMuPDF

Tujuan

Pipeline ini digunakan jika hasil audit Bab 3 menunjukkan:

```
has_text = True
needs_ocr = False
```

Artinya PDF sudah memiliki teks dan tidak perlu OCR.

Script: `scripts/extract_with_pymupdf.py`

```
from pathlib import Path
import fitz

INPUT_FILE = Path("data/pdf/document_001.pdf")
OUTPUT_FILE = Path("data/raw_text/document_001_pymupdf.txt")

def extract_text_from_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    pages_text = []

    for page_number, page in enumerate(doc, start=1):
        text = page.get_text("text", sort=True)
        pages_text.append(f"\n\n--- PAGE {page_number} ---\n\n{text}")

    doc.close()
    return "\n".join(pages_text)

def main():
    OUTPUT_FILE.parent.mkdir(parents=True, exist_ok=True)

    extracted_text = extract_text_from_pdf(INPUT_FILE)
    OUTPUT_FILE.write_text(extracted_text, encoding="utf-8")

    print(f"Hasil ekstraksi tersimpan di: {OUTPUT_FILE}")

if __name__ == "__main__":
    main()
```

Jalankan:

```
python scripts/extract_with_pymupdf.py
```

Output:

```
data/raw_text/document_001_pymupdf.txt
```

Penjelasan Script

- `Path` dipakai untuk mengatur lokasi file.
- `fitz.open()` membuka PDF.
- `enumerate(doc, start=1)` membaca halaman dari nomor 1.
- `page.get_text("text", sort=True)` mengambil teks halaman dan mencoba menyusun urutan baca dari atas-kiri ke bawah-kanan.
- `--- PAGE 1 ---` ditambahkan agar sumber halaman tetap bisa dilacak.
- `write_text()` menyimpan hasil ke file `.txt`.

Catatan kritis: PyMuPDF cepat dan praktis, tetapi tetap harus dicek manual. Jika urutan teks kacau, coba pipeline lain.

4.6 Pipeline B — OCR PDF Scan dengan OCRmyPDF

Tujuan

Pipeline ini digunakan jika hasil audit Bab 3 menunjukkan:

```
has_text = False
needs_ocr = True
```

Artinya PDF kemungkinan scan atau *image-only*.

Perintah OCR

```
ocrmypdf \
-l ind+eng \
--deskew \
--clean \
data/pdf/document_001.pdf \
data/ocr/document_001_ocr.pdf
```

Penjelasan Perintah

- **ocrmypdf** menjalankan OCR untuk PDF.
- **-l ind+eng** memakai bahasa Indonesia dan Inggris.
- **--deskew** membantu meluruskan halaman miring.
- **--clean** membantu membersihkan noise gambar.
- **data/pdf/document_001.pdf** adalah input.
- **data/ocr/document_001_ocr.pdf** adalah output hasil OCR.

OCRmyPDF juga mendokumentasikan fitur seperti menghasilkan PDF/A yang dapat dicari, menempatkan teks OCR di bawah gambar, mempertahankan resolusi gambar asli, serta melakukan *deskew* dan pembersihan jika diminta. ([GitHub](#))

Ekstrak Hasil OCR dengan PyMuPDF

Setelah OCR selesai, jalankan ekstraksi teks dari file hasil OCR.

Buat script:

```
scripts/extract_ocr_result.py
```

Isi:

```
from pathlib import Path
import fitz

INPUT_FILE = Path("data/ocr/document_001_ocr.pdf")
OUTPUT_FILE =
Path("data/raw_text/document_001_ocr_text.txt")

def main():
    doc = fitz.open(INPUT_FILE)
    pages_text = []

    for page_number, page in enumerate(doc, start=1):
        text = page.get_text("text", sort=True)
        pages_text.append(f"\n\n--- PAGE {page_number}
---\n\n{text}")

    doc.close()

    OUTPUT_FILE.parent.mkdir(parents=True,
exist_ok=True)
    OUTPUT_FILE.write_text("\n".join(pages_text),
encoding="utf-8")

    print(f"Hasil OCR text tersimpan di: {OUTPUT_FILE}")

if __name__ == "__main__":
    main()
```

Jalankan:

```
python scripts/extract_ocr_result.py
```

Output:

```
data/raw_text/document_001_ocr_text.txt
```

Catatan penting: OCR memperbaiki PDF scan, tetapi tidak menjamin teks sempurna. Nama, angka, singkatan, dan istilah teknis wajib dicek.

4.7 Pipeline C — Konversi PDF Kompleks ke Markdown dengan Docling

Tujuan

Pipeline ini digunakan untuk PDF yang memiliki:

- heading penting,
- tabel,
- layout kompleks,
- formula,
- gambar,
- struktur dokumen yang ingin dipertahankan,
- kebutuhan output Markdown.

Docling diposisikan sebagai tool modern karena mendukung parsing dokumen dan ekspor ke format yang memudahkan proses AI, RAG, dan *agentic systems*. ([Docling](#))

Perintah Docling

```
docling data/pdf/document_001.pdf --to md --output  
data/markdown/
```

Output yang diharapkan:

```
data/markdown/document_001.md
```

Catatan Praktis

Setelah file Markdown terbentuk, buka dan cek:

- apakah judul terbaca sebagai heading,
- apakah paragraf tidak terpotong aneh,
- apakah tabel masih terbaca,
- apakah urutan baca masuk akal,
- apakah gambar memiliki keterangan,
- apakah bagian penting hilang.

Markdown penting untuk RAG karena lebih mudah dibaca manusia dan lebih mudah diproses di Bab 5–7.

4.8 Ekstraksi Teks dan Tabel dengan pdfplumber

Tujuan

pdfplumber digunakan ketika PDF memiliki tabel yang perlu diperiksa. Tool ini berguna untuk PDF berbasis teks, bukan scan murni.

Script: `scripts/extract_with_pdfplumber.py`

```
from pathlib import Path
import pdfplumber

INPUT_FILE = Path("data/pdf/document_001.pdf")
OUTPUT_TEXT = Path("data/raw_text/document_001_pdfplumber.txt")
OUTPUT_TABLES = Path("data/raw_text/document_001_tables.md")

def main():
    text_parts = []
    table_parts = []

    with pdfplumber.open(INPUT_FILE) as pdf:
        for page_number, page in enumerate(pdf.pages, start=1):
            text = page.extract_text() or ""
            text_parts.append(f"\n\n--- PAGE {page_number}
---\n\n{text}")

            tables = page.extract_tables()
            for table_index, table in enumerate(tables,
start=1):
                table_parts.append(f"\n\n## Page {page_number}
Table {table_index}\n")
                for row in table:
                    clean_row = [cell if cell is not None else
"" for cell in row]
                    table_parts.append("| " + " |
".join(clean_row) + " |")

    OUTPUT_TEXT.parent.mkdir(parents=True, exist_ok=True)
    OUTPUT_TEXT.write_text("\n".join(text_parts),
encoding="utf-8")
    OUTPUT_TABLES.write_text("\n".join(table_parts),
encoding="utf-8")

    print(f"Teks tersimpan di: {OUTPUT_TEXT}")
    print(f"Tabel tersimpan di: {OUTPUT_TABLES}")

if __name__ == "__main__":
    main()
```

Jalankan:

```
python scripts/extract_with_pdfplumber.py
```

Output:

```
data/raw_text/document_001_pdfplumber.txt  
data/raw_text/document_001_tables.md
```

Penjelasan Script

- `pdfplumber.open()` membuka PDF.
- `page.extract_text()` mengambil teks dari halaman.
- `page.extract_tables()` mencoba mengambil tabel.
- Tabel disimpan ke format Markdown sederhana.
- Jika sel tabel kosong, script mengganti `None` menjadi string kosong.

Catatan kritis: hasil tabel harus dicek manual. Tabel adalah salah satu sumber kerusakan corpus paling sering.

4.9 Menjaga Heading, Paragraf, Tabel, dan Urutan Baca

Ekstraksi yang baik bukan hanya mengambil karakter. Yang harus dijaga adalah **struktur makna**.

Heading

Heading membantu RAG memahami konteks. Contoh:

```
# Pengisian KRS
```

```
Mahasiswa wajib mengisi KRS...
```

Lebih baik daripada:

```
Pengisian KRS Mahasiswa wajib mengisi KRS...
```

Paragraf

Paragraf jangan digabung terlalu panjang. Paragraf juga jangan dipotong terlalu pendek.

Buruk:

```
Mahasiswa wajib  
mengisi KRS sebelum  
perkuliahan dimulai
```

Lebih baik:

```
Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.
```

Tabel

Jika tabel gagal di ekstrak, ubah menjadi daftar.

Contoh:

```
## Syarat Pengisian KRS
```

- Mahasiswa harus berstatus aktif.
- Mahasiswa harus login ke sistem akademik.
- KRS harus disetujui Dosen Wali.

Urutan Baca

Untuk PDF dua kolom, cek apakah teks terbaca menyilang. Jika kacau, hasil ekstraksi tidak boleh langsung dipakai sebagai corpus.

Prinsip: lebih baik sedikit manual editing daripada corpus rusak masuk RAG.

4.10 Membandingkan Hasil Ekstraksi Antar-Tool

Setelah menjalankan beberapa pipeline, mahasiswa harus membandingkan hasilnya.

Kriteria Perbandingan

Kriteria	Pertanyaan
Teks terbaca	Apakah isi utama muncul?
Urutan baca	Apakah paragraf berurutan?
Heading	Apakah struktur judul/subjudul terlihat?
Tabel	Apakah tabel masih bermakna?
Noise	Apakah header-footer terlalu mengganggu?
Halaman	Apakah nomor halaman bisa dilacak?
Kesiapan RAG	Apakah hasil ini layak dibersihkan dan di- <i>chunk</i> ?

Script: `scripts/compare_extraction_outputs.py`

```
from pathlib import Path
import pandas as pd

FILES_TO_COMPARE = [
    ("pymupdf", Path("data/raw_text/document_001_pymupdf.txt")),
    ("ocrmypdf_pymupdf",
     Path("data/raw_text/document_001_ocr_text.txt")),
    ("pdfplumber", Path("data/raw_text/document_001_pdfplumber.txt")),
    ("docling_markdown", Path("data/markdown/document_001.md")),
]

OUTPUT_FILE = Path("data/metadata/extraction_comparison.csv")

def count_lines(text):
    return len([line for line in text.splitlines() if line.strip()])

def count_markdown_headings(text):
    return len([line for line in text.splitlines() if
                line.strip().startswith("#")])

def main():
    rows = []

    for tool_name, file_path in FILES_TO_COMPARE:
        if not file_path.exists():
            rows.append({
                "tool": tool_name,
                "file": str(file_path),
```

```

        "exists": False,
        "characters": 0,
        "lines": 0,
        "markdown_headings": 0,
        "notes": "File tidak ditemukan",
    })
    continue

    text = file_path.read_text(encoding="utf-8", errors="ignore")

    rows.append({
        "tool": tool_name,
        "file": str(file_path),
        "exists": True,
        "characters": len(text),
        "lines": count_lines(text),
        "markdown_headings": count_markdown_headings(text),
        "notes": "Perlu cek manual",
    })

    OUTPUT_FILE.parent.mkdir(parents=True, exist_ok=True)
    df = pd.DataFrame(rows)
    df.to_csv(OUTPUT_FILE, index=False, encoding="utf-8")

    print(f"Laporan perbandingan tersimpan di: {OUTPUT_FILE}")

if __name__ == "__main__":
    main()

```

Jalankan:

```
python scripts/compare_extraction_outputs.py
```

Output:

```
data/metadata/extraction_comparison.csv
```

Contoh isi:

```

tool,file,exists,characters,lines,markdown_headings,notes
pymupdf,data/raw_text/document_001_pymupdf.txt,True,1842
0,420,0,Perlu cek manual
pdfplumber,data/raw_text/document_001_pdfplumber.txt,True,17890,398,0,Perlu cek manual
docling_markdown,data/markdown/document_001.md,True,1920
0,460,26,Perlu cek manual

```

4.11 Menentukan Hasil Terbaik untuk Corpus RAG

Setelah hasil dibandingkan, pilih satu output utama untuk masuk Bab 5.

Rekomendasi Keputusan

Kondisi PDF	Output Utama yang Disarankan
PDF teks sederhana	PyMuPDF
PDF teks dengan tabel	pdfplumber + cek manual
PDF scan	OCRmyPDF → PyMuPDF
PDF kompleks	Docling Markdown
PDF dua kolom	Cek manual, bandingkan PyMuPDF dan Docling
PDF tabel penting	pdfplumber + konversi tabel ke Markdown

Aturan Praktis

- Jika hasil Docling menjaga heading dengan baik, gunakan Markdown Docling.
- Jika PyMuPDF lebih bersih danurut, gunakan hasil PyMuPDF.
- Jika PDF scan, jangan pakai hasil ekstraksi sebelum OCR.
- Jika tabel penting rusak, buat representasi tabel manual di Markdown.
- Jika semua hasil buruk, PDF perlu diproses ulang atau diganti.

Keputusan terbaik bukan tool paling canggih, tetapi hasil yang paling layak menjadi corpus.

4.12 Menyimpan Hasil Mentah dan Markdown

Jangan menimpa output. Simpan semua hasil per pipeline.

Struktur output Bab 4:

```
data/
├── ocr/
│   └── document_001_ocr.pdf
├── raw_text/
│   ├── document_001_pymupdf.txt
│   ├── document_001_ocr_text.txt
│   ├── document_001_pdfplumber.txt
│   └── document_001_tables.md
├── markdown/
│   └── document_001.md
└── metadata/
    └── extraction_comparison.csv
```

Kenapa hasil mentah perlu disimpan?

- Untuk audit.
- Untuk membandingkan tool.
- Untuk debugging.
- Untuk membuktikan proses.
- Untuk memperbaiki corpus jika hasil akhir buruk.

Corpus yang baik harus punya jejak proses.

4.13 Praktik Utama Bab 4

Tugas Praktik

Mahasiswa menjalankan tiga pipeline:

1. **Pipeline A** — PDF *searchable* dengan PyMuPDF.
2. **Pipeline B** — PDF scan dengan OCRmyPDF lalu ekstraksi teks.
3. **Pipeline C** — PDF kompleks ke Markdown dengan Docling.
4. Tambahan: ekstraksi tabel dengan pdfplumber.
5. Membuat laporan perbandingan ekstraksi.

Output Wajib

```
data/raw_text/document_001_pymupdf.txt
data/raw_text/document_001_pdfplumber.txt
data/markdown/document_001.md
data/metadata/extraction_comparison.csv
```

Output Tambahan

```
data/ocr/document_001_ocr.pdf
data/raw_text/document_001_ocr_text.txt
data/raw_text/document_001_tables.md
```

4.14 Checklist Validasi Hasil Ekstraksi

Sebelum masuk Bab 5, cek:

- Apakah teks utama muncul?
- Apakah urutan baca masuk akal?
- Apakah heading masih terlihat?
- Apakah tabel penting tidak rusak total?
- Apakah nomor halaman masih bisa dilacak?
- Apakah ada header-footer berulang?
- Apakah ada karakter aneh akibat OCR?
- Apakah file Markdown cukup rapi untuk dibersihkan?

Jika banyak jawaban “tidak”, jangan lanjut ke cleaning. Ulangi ekstraksi dengan pipeline lain.

4.15 Troubleshooting Ringkas

Masalah	Penyebab Umum	Solusi
Hasil teks kosong	PDF scan	Jalankan OCRmyPDF
Teks berantakan	Layout kompleks atau dua kolom	Coba Docling atau cek manual
Tabel rusak	Struktur tabel tidak terbaca	Coba pdfplumber, lalu rapikan Markdown
OCR salah baca	Scan buram atau bahasa OCR salah	Gunakan <code>ind+eng</code> , cek kualitas scan
Heading hilang	Tool hanya ekstrak plain text	Gunakan Docling atau tambahkan heading manual
File Markdown terlalu ramai	Banyak artefak layout	Bersihkan di Bab 5
Karakter aneh muncul	OCR noise atau encoding	Bersihkan dengan script cleaning

4.16 Ringkasan Bab 4

Bab ini mengajarkan bahwa ekstraksi PDF adalah proses strategis, bukan sekadar menjalankan satu perintah. Mahasiswa belajar memakai PyMuPDF untuk PDF *searchable*, pdfplumber untuk teks dan tabel, OCRmyPDF untuk PDF scan, dan Docling untuk konversi PDF kompleks ke Markdown. Hasil dari beberapa tool dibandingkan, lalu dipilih yang paling layak menjadi bahan corpus RAG.

Kalimat kunci Bab 4:

Ekstraksi yang benar bukan yang paling cepat, tetapi yang paling menjaga makna, struktur, dan jejak sumber.

BAB 5 — Cleaning, Normalisasi, dan Metadata Corpus

Corpus yang kotor akan membuat *retrieval* kotor; *retrieval* yang kotor akan membuat jawaban LLM ikut kotor.

5.1 Dari Teks Mentah ke Corpus Bersih

Bab 4 menghasilkan teks mentah dan Markdown dari PDF. Namun, hasil ekstraksi belum otomatis layak menjadi *corpus* RAG. Teks masih bisa mengandung:

- header dan footer berulang,
- nomor halaman,
- watermark,
- karakter aneh hasil OCR,
- spasi berlebihan,
- paragraf pecah,
- heading hilang,
- urutan teks tidak rapi.

Bab 5 mengubah hasil ekstraksi menjadi **clean corpus**. Dalam outline buku, output Bab 5 adalah `clean/document_001.clean.md` dan `metadata/corpus_index.csv`. Ini sejalan dengan arah buku: membuat corpus yang bersih, berstruktur, ber-metadata, dan siap masuk tahap *chunking*.

Pesan utama bab ini: jangan langsung membuat *chunk* dari teks mentah. Bersihkan dulu.

5.2 Peta Halaman Bab 5

Bagian	Fokus	Estimasi
5.1–5.3	Konsep raw text vs clean corpus	2 hlm
5.4–5.6	Cleaning noise, OCR, header-footer	3 hlm
5.7–5.8	Normalisasi paragraf, heading, struktur	2 hlm
5.9–5.10	Metadata dan YAML <i>front matter</i>	2 hlm
5.11–5.14	Praktik Python dan corpus index	3 hlm
Total		12 hlm

5.3 Raw Text vs Clean Corpus

Raw text adalah teks hasil ekstraksi langsung dari PDF. Ia belum tentu rapi, belum tentu benar, dan belum tentu siap untuk RAG.

Contoh *raw text*:

```
Panduan Akademik Mahasiswa  
2026
```

```
12
```

```
Pengisian KRS  
Mahasiswa wajib  
mengisi KRS sebelum  
perkuliahan dimulai.
```

```
Panduan Akademik Mahasiswa
```

Masalahnya:

- nomor halaman 12 masuk ke isi,
- header berulang,
- paragraf pecah,
- kalimat terpotong,
- konteks belum diberi metadata.

Clean corpus adalah teks yang sudah dibersihkan, dinormalisasi, diberi struktur, dan diberi metadata.

Contoh *clean corpus*:

```
---  
document_id: "DOC001"  
title: "Panduan Akademik Mahasiswa"  
source_file: "panduan-akademik.pdf"  
year: 2026  
language: "id"  
document_type: "panduan"  
extraction_tool: "docling"  
pages: 32  
status: "clean"  
---  
  
# Panduan Akademik Mahasiswa  
  
## Pengisian KRS  
  
Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.
```

Perbedaannya tajam: **raw text** hanya hasil ekstraksi; **clean corpus** adalah bahan pengetahuan.

5.4 Menghapus Header, Footer, Nomor Halaman, dan Watermark

PDF akademik, SOP, dan laporan biasanya memiliki teks berulang di setiap halaman.

Contoh:

```
Panduan Akademik Mahasiswa 2026
Universitas Contoh
Halaman 12 dari 32
```

Jika teks ini ikut masuk ke setiap *chunk*, RAG akan terganggu. Sistem pencarian bisa menganggap header sebagai informasi penting padahal hanya noise.

Jenis noise yang harus dihapus:

- **header berulang,**
- **footer berulang,**
- **nomor halaman tunggal,**
- **watermark,**
- **teks legal berulang,**
- **garis pemisah berlebihan,**
- **karakter hasil OCR yang rusak.**

Python menyediakan modul `re` untuk *regular expression*, yaitu pencocokan pola teks yang dapat digunakan untuk membersihkan teks seperti nomor halaman, spasi berulang, dan pola header-footer tertentu. Dokumentasi resmi Python menjelaskan bahwa modul `re` mendukung operasi pencarian dan penggantian pola pada string Unicode. ([Python documentation](#))

Catatan kritis: jangan terlalu agresif menghapus. Cleaning yang terlalu keras bisa menghapus isi penting.

5.5 Noise OCR yang Sering Muncul

OCR pada PDF scan bisa menghasilkan kesalahan kecil tetapi berbahaya.

Contoh:

Hasil OCR Salah	Kemungkinan Benar
K R S	KRS
Do sen Wali	Dosen Wali
Mahas1swa	Mahasiswa
Indonesia	Indonesia
Ørganisasi	Organisasi

Untuk corpus RAG, kesalahan seperti ini penting karena pencarian semantik dan pencarian kata kunci bisa terganggu.

Strategi aman:

- koreksi otomatis untuk pola yang jelas,
- koreksi manual untuk istilah penting,
- simpan daftar istilah domain,
- jangan mengubah semua kata secara membabi buta,
- catat perubahan penting dalam metadata.

Prinsipnya: OCR boleh otomatis, tetapi validasi tetap manusia.

5.6 Normalisasi Spasi, Tanda Baca, dan Paragraf

Normalisasi membuat teks lebih konsisten.

Yang perlu dinormalisasi:

- spasi ganda,
- baris kosong berlebihan,
- kata terpotong karena line break,
- paragraf yang pecah,
- tanda baca yang terlalu rapat,
- karakter asing yang tidak diperlukan.

Contoh buruk:

```
Mahasiswa      wajib mengisi  
KRS sebelum perkuliahan  
dimulai .
```

Contoh bersih:

```
Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.
```

Namun, jangan semua line break dihapus. Heading dan daftar tetap harus dijaga.

Tujuan normalisasi bukan membuat teks pendek, tetapi membuat teks stabil.

5.7 Menjaga Struktur Dokumen agar Mudah di-*Chunk*

Bab 6 akan membuat *chunk*. Kualitas *chunk* sangat ditentukan oleh struktur Bab 5.

Struktur yang harus dijaga:

- judul dokumen,
- heading bab,
- subheading,
- paragraf,
- daftar,
- tabel sederhana,
- keterangan gambar,
- sumber halaman jika tersedia.

Contoh struktur bagus:

```
# Panduan Akademik Mahasiswa

## Pengisian KRS

Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.

## Persetujuan Dosen Wali

KRS harus disetujui oleh Dosen Wali.
```

Struktur ini memudahkan chunking berbasis heading.

Contoh struktur buruk:

```
Panduan Akademik Mahasiswa Pengisian KRS Mahasiswa wajib
mengisi KRS sebelum perkuliahan dimulai Persetujuan Dosen
Wali KRS harus disetujui oleh Dosen Wali
```

Struktur buruk akan membuat RAG sulit menemukan konteks.

Corpus yang mudah di-*chunk* adalah corpus yang menjaga struktur makna.

5.8 Metadata Dokumen

Metadata adalah identitas dokumen. Tanpa metadata, corpus sulit diaudit.

Metadata minimal:

Metadata	Fungsi
<code>document_id</code>	ID unik dokumen
<code>title</code>	Judul dokumen
<code>source_file</code>	Nama PDF sumber
<code>year</code>	Tahun dokumen
<code>language</code>	Bahasa dokumen
<code>document_type</code>	Jenis dokumen
<code>extraction_tool</code>	Tool ekstraksi
<code>pages</code>	Jumlah halaman
<code>status</code>	Status corpus

Contoh:

```
document_id: "DOC001"
title: "Panduan Akademik Mahasiswa"
source_file: "panduan-akademik.pdf"
year: 2026
language: "id"
document_type: "panduan"
extraction_tool: "docling"
pages: 32
status: "clean"
```

Metadata membuat corpus bisa:

- dicari,
- disaring,
- diaudit,
- diperbarui,
- diuji,
- dipertanggungjawabkan.

5.9 YAML *Front Matter* untuk Markdown

YAML *front matter* adalah blok metadata di bagian paling atas file Markdown. Format umum dimulai dan diakhiri dengan tiga garis `---`. Dokumentasi Jekyll menyebut *front matter* harus berada di awal file dan berbentuk YAML valid di antara tiga garis pembatas. ([Jekyll](#))

Template corpus:

```
---
document_id: "DOC001"
title: "Panduan Akademik Mahasiswa"
source_file: "panduan-akademik.pdf"
year: 2026
language: "id"
document_type: "panduan"
extraction_tool: "docling"
pages: 32
status: "clean"
---

# Panduan Akademik Mahasiswa

Isi dokumen bersih...
```

Kenapa YAML *front matter* penting?

- metadata menyatu dengan isi,
- mudah dibaca manusia,
- mudah diproses Python,
- cocok untuk Markdown,
- cocok untuk workflow RAG,
- membantu *traceability*.

Catatan: jangan memasukkan data pribadi atau informasi rahasia ke metadata.

5.10 Corpus Index

Selain metadata di setiap file, kita perlu membuat satu file index:

```
metadata/corpus_index.csv
```

Fungsi `corpus_index.csv`:

- daftar semua dokumen,
- status cleaning,
- tool ekstraksi,
- jumlah halaman,
- lokasi file bersih,
- catatan kualitas.

Kolom yang disarankan:

```
document_id,title,source_file,clean_file,year,language,document_type,extraction_tool,pages,status,notes
```

Contoh:

```
document_id,title,source_file,clean_file,year,language,document_type,extraction_tool,pages,status,notes
DOC001,Panduan Akademik
Mahasiswa,panduan-akademik.pdf,clean/document_001.clean.md,2026,id,panduan,docling,32,clean,Siap untuk chunking
```

Corpus index adalah peta kendali proyek. Tanpa index, proyek cepat berantakan saat jumlah PDF bertambah.

5.11 Praktik Python: Script Cleaning Corpus

Tujuan Praktik

Mahasiswa membuat script:

```
scripts/clean_markdown_corpus.py
```

Script ini membaca:

```
data/markdown/document_001.md
```

Lalu menghasilkan:

```
data/clean/document_001.clean.md  
data/metadata/corpus_index.csv
```

Script: `scripts/clean_markdown_corpus.py`

```
from pathlib import Path  
import re  
import pandas as pd  
  
DOCUMENT_ID = "DOC001"  
TITLE = "Panduan Akademik Mahasiswa"  
SOURCE_FILE = "panduan-akademik.pdf"  
YEAR = 2026  
LANGUAGE = "id"  
DOCUMENT_TYPE = "panduan"  
EXTRACTION_TOOL = "docling"  
PAGES = 32  
  
INPUT_FILE = Path("data/markdown/document_001.md")  
CLEAN_FILE = Path("data/clean/document_001.clean.md")  
INDEX_FILE = Path("data/metadata/corpus_index.csv")  
  
def remove_page_markers(text):  
    text = re.sub(r"\n\s*---\s*PAGE\s+\d+\s*---\s*\n", "\n", text,  
flags=re.IGNORECASE)  
    text = re.sub(r"\n\s*\d+\s*\n", "\n", text)  
    return text  
  
def remove_repeated_headers(text):  
    patterns = [  
        r"Panduan Akademik Mahasiswa\s+2026",  
        r"Universitas Contoh",  
        r"Halaman\s+\d+\s+dari\s+\d+",  
    ]
```

```

    ]

    for pattern in patterns:
        text = re.sub(pattern, "", text, flags=re.IGNORECASE)

    return text

def normalize_whitespace(text):
    text = text.replace("\r\n", "\n")
    text = re.sub(r"[ \t]+", " ", text)
    text = re.sub(r"\n{3,}", "\n\n", text)
    text = re.sub(r" +\n", "\n", text)
    return text.strip()

def fix_common_ocr_errors(text):
    replacements = {
        "K R S": "KRS",
        "Do sen Wali": "Dosen Wali",
        "Mahaslswa": "Mahasiswa",
        "lndonesia": "Indonesia",
        "Organisasi": "Organisasi",
    }

    for wrong, correct in replacements.items():
        text = text.replace(wrong, correct)

    return text

def build_front_matter():
    return f"""---
document_id: "{DOCUMENT_ID}"
title: "{TITLE}"
source_file: "{SOURCE_FILE}"
year: {YEAR}
language: "{LANGUAGE}"
document_type: "{DOCUMENT_TYPE}"
extraction_tool: "{EXTRACTION_TOOL}"
pages: {PAGES}
status: "clean"
---

"""

def clean_text(text):
    text = remove_page_markers(text)
    text = remove_repeated_headers(text)
    text = fix_common_ocr_errors(text)
    text = normalize_whitespace(text)
    return text

```

```

def create_corpus_index():
    data = [{
        "document_id": DOCUMENT_ID,
        "title": TITLE,
        "source_file": SOURCE_FILE,
        "clean_file": str(CLEAN_FILE),
        "year": YEAR,
        "language": LANGUAGE,
        "document_type": DOCUMENT_TYPE,
        "extraction_tool": EXTRACTION_TOOL,
        "pages": PAGES,
        "status": "clean",
        "notes": "Siap untuk chunking",
    }]

    df = pd.DataFrame(data)
    INDEX_FILE.parent.mkdir(parents=True, exist_ok=True)
    df.to_csv(INDEX_FILE, index=False, encoding="utf-8")

def main():
    if not INPUT_FILE.exists():
        raise FileNotFoundError(f"File input tidak ditemukan: {INPUT_FILE}")

    raw_text = INPUT_FILE.read_text(encoding="utf-8", errors="ignore")
    cleaned_text = clean_text(raw_text)

    final_text = build_front_matter() + cleaned_text

    CLEAN_FILE.parent.mkdir(parents=True, exist_ok=True)
    CLEAN_FILE.write_text(final_text, encoding="utf-8")

    create_corpus_index()

    print(f"Clean corpus berhasil dibuat: {CLEAN_FILE}")
    print(f"Corpus index berhasil dibuat: {INDEX_FILE}")

if __name__ == "__main__":
    main()

```

5.12 Penjelasan Script

Bagian 1 — Import Library

```
from pathlib import Path
import re
import pandas as pd
```

Penjelasan:

- **Path** dipakai untuk mengelola file dan folder.
- **re** dipakai untuk membersihkan teks memakai pola *regular expression*.
- **pandas** dipakai untuk membuat **corpus_index.csv**.

Modul **re** resmi digunakan untuk operasi pencocokan pola dan penggantian string di Python. ([Python documentation](#))

Bagian 2 — Metadata Dokumen

```
DOCUMENT_ID = "DOC001"
TITLE = "Panduan Akademik Mahasiswa"
SOURCE_FILE = "panduan-akademik.pdf"
YEAR = 2026
LANGUAGE = "id"
DOCUMENT_TYPE = "panduan"
EXTRACTION_TOOL = "docling"
PAGES = 32
```

Bagian ini menyimpan identitas dokumen.

Ubah sesuai dokumen yang dipakai mahasiswa.

Contoh:

- **DOCUMENT_ID** untuk ID unik.
- **TITLE** untuk judul dokumen.
- **SOURCE_FILE** untuk PDF asal.
- **EXTRACTION_TOOL** untuk tool ekstraksi.
- **PAGES** untuk jumlah halaman.

Bagian 3 — Lokasi File

```
INPUT_FILE = Path("data/markdown/document_001.md")
CLEAN_FILE = Path("data/clean/document_001.clean.md")
INDEX_FILE = Path("data/metadata/corpus_index.csv")
```

Penjelasan:

- **INPUT_FILE** adalah file Markdown hasil Bab 4.
- **CLEAN_FILE** adalah output corpus bersih.
- **INDEX_FILE** adalah daftar index corpus.

Bagian 4 — Menghapus Page Marker dan Nomor Halaman

```
def remove_page_markers(text):
    text = re.sub(r"\n\s*---\s*PAGE\s+\d+\s*---\s*\n",
"\n", text, flags=re.IGNORECASE)
    text = re.sub(r"\n\s*\d+\s*\n", "\n", text)
    return text
```

Fungsi ini menghapus:

- marker seperti **--- PAGE 1 ---**,
- baris yang hanya berisi angka halaman.

Catatan kritis: jika angka halaman penting untuk sitasi, jangan dihapus total. Simpan nomor halaman di metadata chunk pada Bab 6.

Bagian 5 — Menghapus Header Berulang

```
def remove_repeated_headers(text):
    patterns = [
        r"Panduan Akademik Mahasiswa\s+2026",
        r"Universitas Contoh",
        r"Halaman\s+\d+\s+dari\s+\d+",
    ]
```

Bagian ini berisi pola header-footer yang ingin dibersihkan.

Contoh:

- judul dokumen yang muncul di setiap halaman,
- nama institusi,
- teks “Halaman 1 dari 32”.

Penting: daftar **patterns** harus disesuaikan untuk setiap PDF.

Bagian 6 — Normalisasi Spasi

```
def normalize_whitespace(text):
    text = text.replace("\r\n", "\n")
    text = re.sub(r"[ \t]+", " ", text)
    text = re.sub(r"\n{3,}", "\n\n", text)
    text = re.sub(r" +\n", "\n", text)
    return text.strip()
```

Fungsi ini:

- menyamakan format baris,
- menghapus spasi berulang,
- membatasi baris kosong berlebihan,
- membersihkan spasi sebelum akhir baris.

Hasilnya lebih rapi dan lebih siap di-*chunk*.

Bagian 7 — Koreksi OCR Umum

```
def fix_common_ocr_errors(text):
    replacements = {
        "K R S": "KRS",
        "Do sen Wali": "Dosen Wali",
        "Mahas1swa": "Mahasiswa",
        "lndonesia": "Indonesia",
        "0rganisasi": "Organisasi",
    }
```

Fungsi ini memperbaiki kesalahan OCR yang sering muncul.

Strategi aman:

- koreksi hanya kata yang jelas salah,
- jangan menebak istilah teknis,
- catat istilah penting dalam daftar koreksi,
- validasi manual untuk istilah domain.

Bagian 8 — Membuat YAML *Front Matter*

```
def build_front_matter():
    return f"""---
document_id: "{DOCUMENT_ID}"
title: "{TITLE}"
source_file: "{SOURCE_FILE}"
year: {YEAR}
language: "{LANGUAGE}"
document_type: "{DOCUMENT_TYPE}"
extraction_tool: "{EXTRACTION_TOOL}"
pages: {PAGES}
status: "clean"
---

"""
```

Fungsi ini membuat blok metadata di awal file Markdown.

YAML *front matter* umum digunakan untuk menyimpan metadata di awal dokumen Markdown. Dokumentasi Jekyll menegaskan bahwa *front matter* berada di awal file dan berbentuk YAML valid di antara tiga garis pembatas. ([Jekyll](#))

Bagian 9 — Membuat Corpus Index

```
def create_corpus_index():
    data = [{
        "document_id": DOCUMENT_ID,
        "title": TITLE,
        "source_file": SOURCE_FILE,
        "clean_file": str(CLEAN_FILE),
        "year": YEAR,
        "language": LANGUAGE,
        "document_type": DOCUMENT_TYPE,
        "extraction_tool": EXTRACTION_TOOL,
        "pages": PAGES,
        "status": "clean",
        "notes": "Siap untuk chunking",
    }]
```

Fungsi ini membuat satu baris metadata dokumen.

Lalu disimpan sebagai CSV:

```
df.to_csv(INDEX_FILE, index=False, encoding="utf-8")
```

CSV ini menjadi peta kendali seluruh corpus.

5.13 Menjalankan Script

Pastikan file input ada:

```
data/markdown/document_001.md
```

Jalankan:

```
python scripts/clean_markdown_corpus.py
```

Output:

```
data/clean/document_001.clean.md  
data/metadata/corpus_index.csv
```

Contoh pesan berhasil:

```
Clean corpus berhasil dibuat:  
data/clean/document_001.clean.md  
Corpus index berhasil dibuat: data/metadata/corpus_index.csv
```

5.14 Contoh Output Clean Corpus

```
---
document_id: "DOC001"
title: "Panduan Akademik Mahasiswa"
source_file: "panduan-akademik.pdf"
year: 2026
language: "id"
document_type: "panduan"
extraction_tool: "docling"
pages: 32
status: "clean"
---

# Panduan Akademik Mahasiswa

## Pengisian KRS

Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.

## Persetujuan Dosen Wali

KRS harus disetujui oleh Dosen Wali sebelum mahasiswa
mengikuti perkuliahan.
```

Ciri output bagus:

- metadata jelas,
- heading rapi,
- paragraf bersih,
- noise hilang,
- kalimat mudah dibaca,
- siap masuk Bab 6.

5.15 Contoh Output Corpus Index

```
document_id,title,source_file,clean_file,year,language,document_type,extraction_tool,pages,status,notes
DOC001,Panduan Akademik
Mahasiswa,panduan-akademik.pdf,data/clean/document_001.clean.md,2026,id,panduan,docling,32,clean,Siap untuk chunking
```

Index ini penting saat jumlah dokumen bertambah.

Contoh nanti:

```
DOC001 → Panduan Akademik
DOC002 → SOP Laboratorium
DOC003 → Brosur Layanan Kampus
```

Tanpa index, corpus akan sulit dikelola.

5.16 Versioning Corpus dengan Git

Corpus adalah aset yang terus berubah. Hari ini dibersihkan, besok diperbaiki, minggu depan ditambah dokumen baru.

Git digunakan untuk mencatat perubahan. Buku resmi Git menjelaskan bahwa sistem version control membantu melacak perubahan file dari waktu ke waktu dan memungkinkan pengguna melihat atau mengembalikan versi sebelumnya. ([Git](#))

Perintah dasar:

```
git init
git add README.md scripts/ data/clean/ data/metadata/
git commit -m "Add clean corpus and corpus index"
```

Penjelasan:

- **git init** membuat repository.
- **git add** memilih file yang akan dicatat.
- **git commit** menyimpan snapshot perubahan.
- **-m** menambahkan pesan commit.

Jangan masukkan file besar sembarangan. PDF mentah yang sangat besar bisa dikelola terpisah sesuai kebutuhan proyek.

5.17 Checklist Validasi Cleaning

Sebelum masuk Bab 6, cek:

- Apakah teks utama masih lengkap?
- Apakah header-footer berulang sudah hilang?
- Apakah nomor halaman yang tidak perlu sudah bersih?
- Apakah paragraf mudah dibaca?
- Apakah heading masih terlihat?
- Apakah istilah penting tidak ikut terhapus?
- Apakah metadata sudah benar?
- Apakah file clean corpus tersimpan di folder yang benar?
- Apakah corpus index sudah dibuat?

Jika banyak jawaban “tidak”, ulangi cleaning.

5.18 Troubleshooting Ringkas

Masalah	Penyebab Umum	Solusi
Isi penting hilang	Regex terlalu agresif	Kurangi pola penghapusan
Header masih muncul	Pola header belum lengkap	Tambahkan pattern baru
Teks masih berantakan	Hasil ekstraksi Bab 4 buruk	Coba output tool lain
Paragraf terlalu panjang	Line break dihapus berlebihan	Pertahankan pemisah paragraf
Metadata salah	Input manual belum diperbarui	Cek variabel metadata
CSV kosong	File input tidak terbaca	Cek lokasi file
Karakter aneh masih muncul	OCR buruk	Perbaiki daftar koreksi OCR

Prinsip debugging: cleaning yang baik harus bisa dijelaskan, bukan sekadar “kelihatan rapi”.

5.19 Praktik Utama Bab 5

Tugas Praktik

Mahasiswa harus:

1. Membaca file Markdown hasil Bab 4.
2. Menghapus noise utama.
3. Menormalkan spasi dan paragraf.
4. Menambahkan YAML *front matter*.
5. Membuat file clean corpus.
6. Membuat corpus index.
7. Mencatat versi perubahan dengan Git.

Output Wajib

```
data/clean/document_001.clean.md
data/metadata/corpus_index.csv
```

Output Tambahan

```
requirements.txt
output/cleaning_notes.md
```

Contoh isi `output/cleaning_notes.md`:

```
# Cleaning Notes

## DOC001
Sumber: panduan-akademik.pdf

Perubahan:
- Menghapus header berulang.
- Menghapus nomor halaman tunggal.
- Normalisasi spasi.
- Koreksi OCR: K R S → KRS.
- Menambahkan YAML front matter.

Status:
Siap untuk Bab 6 – Chunking Corpus.
```

Ringkasan Bab 5

Bab ini mengubah hasil ekstraksi mentah menjadi **clean corpus**. Mahasiswa belajar membedakan *raw text* dan *clean corpus*, menghapus noise, menormalkan teks, menjaga struktur, menambahkan metadata, membuat YAML *front matter*, dan membangun `corpus_index.csv`.

Kalimat kunci Bab 5:

Corpus yang bersih bukan hanya enak dibaca, tetapi membuat RAG lebih presisi, lebih dapat diaudit, dan lebih layak dipercaya.

BAB 6 — Chunking Corpus agar Mudah Dicari LLM

Chunking bukan memotong teks agar pendek; chunking adalah menjaga makna agar LLM menemukan konteks yang benar.

6.1 Mengapa Chunking Menentukan Kualitas RAG

Setelah Bab 5 menghasilkan *clean corpus*, langkah berikutnya adalah membuat *chunk*. Dalam RAG, *chunk* adalah potongan teks yang akan dicari oleh sistem sebelum LLM membuat jawaban.

Jika *chunk* buruk, sistem bisa mengambil konteks yang salah. Jika konteks salah, jawaban LLM ikut salah.

Outline buku menegaskan bahwa *chunk* RAG harus menyimpan `chunk_id`, `document_id`, `title`, `section`, `page_start`, `page_end`, dan `text`, agar setiap potongan teks tetap memiliki konteks dan jejak sumber.

Pesan utama:

Chunk yang baik membuat RAG lebih presisi, lebih mudah diaudit, dan lebih aman dari halusinasi.

6.2 Peta Halaman Bab 6

Bagian	Fokus	Estimasi
6.1–6.3	Konsep chunking dan risiko chunk buruk	2 hlm
6.4–6.6	Strategi chunk: heading, paragraf, halaman, overlap	3 hlm
6.7–6.8	Metadata chunk dan evaluasi kualitas	2 hlm
6.9–6.13	Praktik Python membuat JSONL dan laporan chunk	4 hlm
6.14–6.16	Validasi, troubleshooting, ringkasan	1 hlm
Total		12 hlm

6.3 Apa Itu Chunk dalam RAG

Chunk adalah potongan kecil dari corpus yang disimpan agar mudah dicari.

Contoh dokumen panjang:

Panduan Akademik Mahasiswa berisi aturan KRS, KHS, cuti akademik, pembayaran, dan wisuda.

Dalam RAG, dokumen panjang dipotong menjadi bagian lebih kecil:

Chunk 1: aturan KRS
Chunk 2: aturan KHS
Chunk 3: cuti akademik
Chunk 4: pembayaran
Chunk 5: wisuda

Tujuannya:

- pencarian lebih fokus,
- konteks lebih pendek,
- jawaban lebih relevan,
- sumber lebih mudah dilacak.

Open WebUI menyediakan pengaturan *chunk size* dan *chunk overlap* untuk dokumen RAG; dokumentasinya juga menyebutkan bahwa dokumen lama perlu *re-index* jika pengaturan chunk berubah agar kualitas retrieval konsisten. ([Open WebUI](#))

6.4 Chunk Terlalu Kecil vs Chunk Terlalu Besar

Chunking adalah keseimbangan.

Chunk Terlalu Kecil

Contoh:

`Mahasiswa wajib.`

Masalah:

- konteks hilang,
- subjek tidak jelas,
- sulit menjawab pertanyaan,
- rawan salah interpretasi.

Chunk Terlalu Besar

Contoh:

`Seluruh Bab Akademik sepanjang 10 halaman dimasukkan dalam satu chunk.`

Masalah:

- pencarian kurang presisi,
- terlalu banyak informasi tidak relevan,
- jawaban bisa melebar,
- biaya konteks lebih besar,
- bagian penting bisa tenggelam.

LangChain menjelaskan bahwa `chunk_size` menentukan ukuran maksimum chunk, sedangkan `chunk_overlap` menentukan overlap antar chunk untuk mengurangi hilangnya informasi ketika konteks terpotong di batas chunk. ([Dokumen LangChain](#))

Aturan praktis: chunk harus cukup pendek untuk dicari, tetapi cukup panjang untuk menjawab.

6.5 Strategi Chunk Berbasis Heading

Strategi terbaik untuk dokumen terstruktur adalah *heading-based chunking*. Heading membantu menjaga konteks.

Contoh Markdown:

```
# Panduan Akademik Mahasiswa

## Pengisian KRS

Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.

## Persetujuan Dosen Wali

KRS harus disetujui oleh Dosen Wali.
```

Chunk yang baik:

```
{"section": "Pengisian KRS", "text": "Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai."}
```

Kelebihan:

- konteks jelas,
- cocok untuk dokumen akademik,
- cocok untuk SOP,
- mudah ditelusuri,
- mudah dipakai sebagai metadata.

Kekurangan:

- tidak cocok jika dokumen tidak punya heading,
- hasil ekstraksi PDF kadang kehilangan heading,
- heading perlu dirapikan di Bab 5.

Jika dokumen punya heading, gunakan heading sebagai jangkar utama.

6.6 Strategi Chunk Berbasis Paragraf

Jika heading tidak rapi, gunakan paragraf.

Prinsip:

- satu chunk berisi beberapa paragraf terkait,
- jangan memotong kalimat di tengah,
- jangan mencampur dua topik berbeda,
- simpan section jika diketahui.

Contoh:

Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai. KRS harus disetujui oleh Dosen Wali.

Chunk:

```
{  
  "section": "Pengisian KRS",  
  "text": "Mahasiswa wajib mengisi KRS sebelum perkuliahan  
dimulai. KRS harus disetujui oleh Dosen Wali."  
}
```

Kelebihan:

- mudah diterapkan,
- cocok untuk teks naratif,
- menjaga kalimat tetap utuh.

Kekurangan:

- bisa kehilangan struktur dokumen,
- paragraf terlalu panjang perlu dipotong lagi.

6.7 Strategi Chunk Berbasis Halaman

Chunk berbasis halaman berguna jika nomor halaman penting.

Cocok untuk:

- regulasi,
- SOP resmi,
- dokumen kebijakan,
- dokumen akademik,
- dokumen yang harus punya sitasi halaman.

Contoh:

```
{
  "chunk_id": "DOC001-P005",
  "document_id": "DOC001",
  "page_start": 5,
  "page_end": 5,
  "text": "Mahasiswa wajib mengisi KRS..."
}
```

Kelebihan:

- sumber mudah dilacak,
- cocok untuk audit,
- mudah menjelaskan rujukan halaman.

Kekurangan:

- satu halaman bisa berisi banyak topik,
- topik bisa terpotong antar halaman,
- chunk bisa terlalu panjang atau terlalu pendek.

Untuk buku ini, strategi terbaik adalah kombinasi: heading + paragraf + metadata halaman.

6.8 Chunk Overlap

Chunk overlap berarti sebagian teks diulang pada chunk berikutnya. Tujuannya menjaga konteks jika informasi penting terpotong di batas chunk.

Contoh tanpa overlap:

Chunk 1: Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.

Chunk 2: Persetujuan diberikan oleh Dosen Wali.

Masalah: chunk kedua tidak jelas apa yang disetujui.

Contoh dengan overlap:

Chunk 1: Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.

Chunk 2: KRS sebelum perkuliahan dimulai. Persetujuan diberikan oleh Dosen Wali.

Overlap membantu menjaga hubungan antar kalimat.

Dokumentasi LangChain menekankan bahwa overlap membantu mengurangi kehilangan informasi ketika konteks terbagi antar chunk. ([Dokumen LangChain](#))

Aturan praktis awal:

- chunk kecil: overlap kecil,
- chunk panjang: overlap sedang,
- dokumen SOP: overlap secukupnya,
- dokumen tabel: jangan terlalu banyak overlap.

6.9 Metadata per Chunk

Chunk tanpa metadata adalah potongan teks tanpa alamat. Untuk RAG serius, setiap chunk harus punya identitas.

Format wajib:

```
{
  "chunk_id": "DOC001-0001",
  "document_id": "DOC001",
  "title": "Panduan Akademik Mahasiswa",
  "section": "KRS",
  "page_start": 5,
  "page_end": 6,
  "text": "Mahasiswa wajib mengisi KRS..."
}
```

Fungsi metadata:

- melacak sumber,
- memudahkan evaluasi,
- memudahkan debugging,
- memudahkan filtering,
- memudahkan sitasi,
- memudahkan update corpus.

Metadata bukan hiasan. Metadata adalah alat audit.

6.10 Format JSONL untuk Chunk

JSONL adalah format satu baris satu objek JSON.

Contoh:

```
{ "chunk_id": "DOC001-0001", "document_id": "DOC001", "title": "Pan  
duan Akademik  
Mahasiswa", "section": "KRS", "page_start": 5, "page_end": 6, "text":  
"Mahasiswa wajib mengisi KRS..." }  
{ "chunk_id": "DOC001-0002", "document_id": "DOC001", "title": "Pan  
duan Akademik Mahasiswa", "section": "Dosen  
Wali", "page_start": 6, "page_end": 7, "text": "KRS harus disetujui  
oleh Dosen Wali..." }
```

Kelebihan JSONL:

- mudah diproses Python,
- cocok untuk data besar,
- satu chunk satu baris,
- mudah dicek,
- mudah dipakai untuk pipeline ingestion.

Dokumentasi JSON Lines menyebut format ini cocok untuk pemrosesan record satu per baris, dengan setiap baris merupakan nilai JSON valid dan menggunakan encoding UTF-8.

([Dokumen LangChain](#))

6.11 Praktik Python: Membuat Chunk dari Clean Markdown

Tujuan Praktik

Membaca:

```
data/clean/document_001.clean.md
```

Menghasilkan:

```
data/chunks/document_001.chunks.jsonl
data/metadata/chunk_report.csv
```

Script: `scripts/create_chunks.py`

```
from pathlib import Path
import json
import re
import pandas as pd

INPUT_FILE = Path("data/clean/document_001.clean.md")
CHUNK_FILE = Path("data/chunks/document_001.chunks.jsonl")
REPORT_FILE = Path("data/metadata/chunk_report.csv")

DOCUMENT_ID = "DOC001"
TITLE = "Panduan Akademik Mahasiswa"
DEFAULT_PAGE_START = 1
DEFAULT_PAGE_END = 1

MAX_CHARS = 1200
OVERLAP_CHARS = 150

def remove_front_matter(text):
    pattern = r"^---\n.*?\n---\n"
    return re.sub(pattern, "", text, flags=re.DOTALL).strip()

def split_by_heading(text):
    sections = []
    current_heading = "Tanpa Section"
    current_lines = []

    for line in text.splitlines():
        stripped = line.strip()

        if stripped.startswith("#"):
```

```

        if current_lines:
            sections.append({
                "section": current_heading,
                "text": "\n".join(current_lines).strip()
            })
            current_lines = []

            current_heading = stripped.lstrip("#").strip()
        else:
            current_lines.append(line)

    if current_lines:
        sections.append({
            "section": current_heading,
            "text": "\n".join(current_lines).strip()
        })

    return [section for section in sections if
            section["text"]]

def split_long_text(text, max_chars=MAX_CHARS,
                    overlap_chars=OVERLAP_CHARS):
    chunks = []
    start = 0

    while start < len(text):
        end = start + max_chars
        chunk = text[start:end].strip()

        if chunk:
            chunks.append(chunk)

        if end >= len(text):
            break

        start = end - overlap_chars

    return chunks

def create_chunks(sections):
    chunks = []
    chunk_counter = 1

    for section in sections:
        section_name = section["section"]
        section_text = section["text"]

```

```

text_parts = split_long_text(section_text)

for part in text_parts:
    chunk_id = f"{DOCUMENT_ID}-{chunk_counter:04d}"

    chunks.append({
        "chunk_id": chunk_id,
        "document_id": DOCUMENT_ID,
        "title": TITLE,
        "section": section_name,
        "page_start": DEFAULT_PAGE_START,
        "page_end": DEFAULT_PAGE_END,
        "text": part,
    })

    chunk_counter += 1

return chunks

def save_jsonl(chunks, output_file):
    output_file.parent.mkdir(parents=True, exist_ok=True)

    with output_file.open("w", encoding="utf-8") as file:
        for chunk in chunks:
            file.write(json.dumps(chunk, ensure_ascii=False)
+ "\n")

def create_report(chunks):
    rows = []

    for chunk in chunks:
        rows.append({
            "chunk_id": chunk["chunk_id"],
            "document_id": chunk["document_id"],
            "section": chunk["section"],
            "page_start": chunk["page_start"],
            "page_end": chunk["page_end"],
            "char_length": len(chunk["text"]),
            "word_count": len(chunk["text"].split()),
        })

    REPORT_FILE.parent.mkdir(parents=True, exist_ok=True)
    df = pd.DataFrame(rows)
    df.to_csv(REPORT_FILE, index=False, encoding="utf-8")

```

```

def main():
    if not INPUT_FILE.exists():
        raise FileNotFoundError(f"File tidak ditemukan:
{INPUT_FILE}")

    raw_text = INPUT_FILE.read_text(encoding="utf-8",
errors="ignore")
    text_without_metadata = remove_front_matter(raw_text)

    sections = split_by_heading(text_without_metadata)
    chunks = create_chunks(sections)

    save_jsonl(chunks, CHUNK_FILE)
    create_report(chunks)

    print(f"Chunk berhasil dibuat: {CHUNK_FILE}")
    print(f"Laporan chunk berhasil dibuat: {REPORT_FILE}")
    print(f"Jumlah chunk: {len(chunks)}")

if __name__ == "__main__":
    main()

```

6.12 Penjelasan Script

Import Library

```
from pathlib import Path
import json
import re
import pandas as pd
```

Penjelasan:

- `Path` mengelola lokasi file.
- `json` menyimpan chunk ke format JSONL.
- `re` membersihkan YAML *front matter*.
- `pandas` membuat laporan CSV.

Parameter Chunk

```
MAX_CHARS = 1200
OVERLAP_CHARS = 150
```

Penjelasan:

- `MAX_CHARS` adalah panjang maksimum chunk.
- `OVERLAP_CHARS` adalah teks yang diulang antar chunk.

Angka ini bukan angka suci. Untuk dokumen pendek, bisa lebih kecil. Untuk dokumen teknis panjang, bisa lebih besar.

LlamaIndex, misalnya, mendokumentasikan bahwa dokumen yang di-*ingest* ke index dipotong menjadi chunk dengan overlap; pengaturan umum dapat disesuaikan sesuai jenis data dan hasil retrieval yang diinginkan. ([Developer Documentation](#))

Menghapus YAML Front Matter

```
pattern = r"^---\n.*?\n---\n"
```

Bagian ini menghapus metadata di awal file agar tidak ikut menjadi isi chunk.

Split Berdasarkan Heading

```
if stripped.startswith("#"):
```

Script menganggap baris yang dimulai # sebagai heading Markdown.

Contoh:

```
## Pengisian KRS
```

Akan menjadi:

```
section = "Pengisian KRS"
```

Membagi Teks Panjang

```
start = end - overlap_chars
```

Baris ini membuat overlap. Jadi chunk berikutnya mengambil sedikit bagian akhir dari chunk sebelumnya.

Menyimpan JSONL

```
file.write(json.dumps(chunk, ensure_ascii=False) + "\n")
```

`ensure_ascii=False` penting agar teks Indonesia tidak berubah menjadi kode Unicode yang sulit dibaca.

6.13 Menjalankan Script

Pastikan file berikut sudah ada:

```
data/clean/document_001.clean.md
```

Jalankan:

```
python scripts/create_chunks.py
```

Output:

```
data/chunks/document_001.chunks.jsonl
data/metadata/chunk_report.csv
```

Contoh pesan:

```
Chunk berhasil dibuat: data/chunks/document_001.chunks.jsonl
Laporan chunk berhasil dibuat: data/metadata/chunk_report.csv
Jumlah chunk: 18
```

6.14 Contoh Output chunks.jsonl

```
{"chunk_id":"DOC001-0001","document_id":"DOC001","title":"Pan
duan Akademik Mahasiswa","section":"Pengisian
KRS","page_start":1,"page_end":1,"text":"Mahasiswa wajib
mengisi KRS sebelum perkuliahan dimulai. KRS harus disetujui
oleh Dosen Wali."}
{"chunk_id":"DOC001-0002","document_id":"DOC001","title":"Pan
duan Akademik Mahasiswa","section":"Cuti
Akademik","page_start":1,"page_end":1,"text":"Mahasiswa dapat
mengajukan cuti akademik sesuai ketentuan yang berlaku."}
```

Ciri chunk bagus:

- punya ID unik,
- punya judul dokumen,
- punya section,
- punya teks yang utuh,
- tidak terlalu pendek,
- tidak terlalu panjang,
- bisa dilacak.

6.15 Contoh Output `chunk_report.csv`

```
chunk_id,document_id,section,page_start,page_end,char_length,  
word_count  
DOC001-0001,DOC001,Pengisian KRS,1,1,185,24  
DOC001-0002,DOC001,Cuti Akademik,1,1,92,12
```

Laporan ini dipakai untuk mengecek:

- chunk terlalu pendek,
- chunk terlalu panjang,
- section kosong,
- jumlah chunk terlalu banyak,
- distribusi panjang chunk tidak wajar.

6.16 Evaluasi Chunk: Apakah Masih Bisa Menjawab Pertanyaan?

Chunk harus diuji dengan pertanyaan sederhana.

Contoh:

Pertanyaan	Chunk yang Harus Muncul
Siapa yang menyetujui KRS?	Chunk tentang Dosen Wali
Apa isi KRS?	Chunk tentang daftar mata kuliah
Kapan mahasiswa mengisi KRS?	Chunk tentang jadwal KRS
Apa syarat cuti akademik?	Chunk tentang cuti akademik

Jika pertanyaan tidak bisa dijawab dari satu atau dua chunk, berarti chunk mungkin:

- terlalu kecil,
- terlalu besar,
- salah section,
- kehilangan heading,
- kehilangan metadata,
- perlu overlap lebih baik.

Evaluasi chunk harus dilakukan sebelum upload ke RAG.

6.17 Membuat Folder `rag_upload/`

Folder `rag_upload/` adalah tempat file final untuk Bab 7.

Buat folder jika belum ada:

```
mkdir -p data/rag_upload
```

Salin file chunk awal:

```
cp data/chunks/document_001.chunks.jsonl  
data/rag_upload/corpus_all.jsonl
```

Nanti di Bab 7, folder ini akan dilengkapi:

```
data/rag_upload/  
├─ corpus_for_rag.md  
├─ corpus_all.jsonl  
├─ metadata.csv  
└─ upload_readme.md
```

Jangan upload file eksperimen. Upload hanya file final yang sudah dipilih.

6.18 Checklist Validasi Chunk

Sebelum lanjut ke Bab 7, cek:

- Apakah semua chunk punya `chunk_id`?
- Apakah semua chunk punya `document_id`?
- Apakah `section` tidak kosong?
- Apakah teks chunk masih bisa dipahami?
- Apakah chunk tidak terlalu pendek?
- Apakah chunk tidak terlalu panjang?
- Apakah ada overlap yang cukup?
- Apakah metadata sumber masih ada?
- Apakah `chunk_report.csv` sudah dibuat?
- Apakah file JSONL valid?

Jika banyak masalah, kembali ke Bab 5. Jangan memaksa chunk buruk masuk RAG.

6.19 Troubleshooting Ringkas

Masalah	Penyebab Umum	Solusi
Chunk terlalu pendek	Heading terlalu banyak atau paragraf pecah	Gabungkan paragraf kecil
Chunk terlalu panjang	Section terlalu besar	Turunkan <code>MAX_CHARS</code>
Konteks hilang	Overlap terlalu kecil	Naikkan <code>OVERLAP_CHARS</code>
Section kosong	Heading Markdown tidak rapi	Perbaiki heading di Bab 5
JSONL error	Ada format JSON rusak	Gunakan <code>json.dumps()</code>
Banyak duplikasi	Overlap terlalu besar	Kurangi overlap
Jawaban RAG melebar	Chunk terlalu besar	Perkecil chunk
Jawaban RAG tidak lengkap	Chunk terlalu kecil	Perbesar chunk atau tambah overlap

6.20 Praktik Utama Bab 6

Tugas Praktik

Mahasiswa harus:

1. Membaca file `clean/document_001.clean.md`.
2. Menghapus YAML *front matter* dari isi chunk.
3. Memotong teks berdasarkan heading.
4. Memecah section panjang dengan overlap.
5. Menyimpan hasil ke JSONL.
6. Membuat laporan chunk.
7. Menyalin output awal ke folder `rag_upload/`.

Output Wajib

```
data/chunks/document_001.chunks.jsonl
data/metadata/chunk_report.csv
```

Output Tambahan

```
data/rag_upload/corpus_all.jsonl
```

Ringkasan Bab 6

Bab ini mengajarkan bahwa *chunking* bukan pekerjaan mekanis. Chunk yang baik harus menjaga konteks, heading, sumber, dan makna. Mahasiswa belajar membuat chunk berbasis heading, paragraf, halaman, dan overlap, lalu menyimpannya ke format JSONL dengan metadata lengkap.

Kalimat kunci Bab 6:

RAG tidak mencari dokumen; RAG mencari chunk. Jika chunk salah, jawaban juga mudah salah.

BAB 7 — Menyiapkan File Upload RAG: Markdown, JSONL, dan CSV

File kerja belum tentu file upload. RAG membutuhkan paket corpus final yang bersih, konsisten, ber-metadata, dan siap dicari oleh LLM.

7.1 Pembuka Bab — Dari File Kerja ke File Final

Bab 6 sudah menghasilkan *chunk* dalam format JSONL. Namun, proses belum selesai. Banyak mahasiswa berhenti setelah berhasil mengekstrak teks atau membuat *chunk*. Ini belum cukup.

Target buku ini adalah membuat **file corpus siap upload ke RAG LLM**. Karena itu, Bab 7 membedakan dua jenis file:

- **file kerja internal**, yaitu file yang dipakai selama proses ekstraksi, cleaning, dan eksperimen;
- **file final upload**, yaitu file yang sudah dipilih, dirapikan, dan siap masuk ke *Knowledge Base*.

Outline buku menegaskan bahwa output Bab 7 adalah folder `rag_upload/` berisi `corpus_for_rag.md`, `corpus_all.jsonl`, `metadata.csv`, dan `upload_readme.md`.

Intinya: jangan upload semua file. Upload hanya corpus final yang sudah siap dicari.

7.2 Peta Halaman Bab 7

Bagian	Fokus	Estimasi
7.1–7.3	Konsep file kerja vs file upload	2 hlm
7.4–7.6	Format Markdown, JSONL, CSV	3 hlm
7.7–7.11	Praktik membuat folder <code>rag_upload/</code>	4 hlm
7.12–7.15	Validasi, checklist, troubleshooting	1 hlm
Total		10 hlm

7.3 File Kerja Internal vs File Upload RAG

Dalam proyek corpus, file yang dihasilkan sangat banyak.

Contoh file kerja:

```
data/pdf/document_001.pdf
data/ocr/document_001_ocr.pdf
data/raw_text/document_001_pymupdf.txt
data/markdown/document_001.md
data/clean/document_001.clean.md
data/chunks/document_001.chunks.jsonl
data/metadata/chunk_report.csv
```

Semua file ini berguna untuk proses, tetapi tidak semuanya perlu diupload.

File final untuk RAG harus sederhana:

```
data/rag_upload/
├── corpus_for_rag.md
├── corpus_all.jsonl
├── metadata.csv
└── upload_readme.md
```

Prinsip strategis:

- `corpus_for_rag.md` untuk dibaca manusia dan sistem RAG berbasis dokumen.
- `corpus_all.jsonl` untuk pipeline yang membutuhkan *chunk* terstruktur.
- `metadata.csv` untuk audit dan evaluasi.
- `upload_readme.md` untuk menjelaskan isi corpus.

Open WebUI mendukung upload file untuk RAG melalui chat upload atau *Knowledge Base*, dan *Knowledge Base* dipakai untuk koleksi dokumen yang lebih permanen serta terorganisasi. ([Open WebUI](#))

7.4 Format File yang Umum Dipakai untuk RAG

Format yang aman untuk buku ini:

Format	Fungsi	Kelebihan
Markdown .md	Corpus bersih final	Mudah dibaca, mudah diedit, menjaga heading
JSONL .jsonl	Chunk terstruktur	Satu baris satu chunk, mudah diproses Python
CSV .csv	Metadata dan evaluasi	Mudah dibaca sebagai tabel
TXT .txt	Teks sederhana	Cocok untuk dokumen kecil
Turtle .ttl	Ontologi ringan	Dipakai nanti untuk relasi pengetahuan

Bab ini fokus pada tiga format utama: **Markdown**, **JSONL**, **CSV**.

CommonMark menyediakan spesifikasi Markdown yang lebih standar dan tidak ambigu, sehingga Markdown cocok dipakai sebagai format teks terstruktur yang tetap mudah dibaca manusia. ([commonmark.org](#)) JSON Lines cocok untuk data per baris karena setiap baris adalah nilai JSON valid, memakai UTF-8, dan dipisahkan dengan newline. ([jsonlines.org](#))

7.5 Markdown untuk Dokumen Bersih

Markdown dipakai untuk menyimpan corpus yang sudah bersih.

Contoh:

```
---
document_id: "DOC001"
title: "Panduan Akademik Mahasiswa"
source_file: "panduan-akademik.pdf"
year: 2026
language: "id"
document_type: "panduan"
status: "ready_for_rag"
---

# Panduan Akademik Mahasiswa

## Pengisian KRS

Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.

## Persetujuan Dosen Wali

KRS harus disetujui oleh Dosen Wali.
```

Kelebihan Markdown:

- mudah dibaca manusia,
- heading tetap terlihat,
- metadata bisa disimpan di awal file,
- mudah diproses Python,
- cocok untuk RAG berbasis dokumen.

Markdown adalah file utama untuk upload jika sistem RAG menerima dokumen teks.

Catatan penting: Markdown final tidak boleh berisi catatan eksperimen, error log, atau teks mentah yang belum dibersihkan.

7.6 JSONL untuk Chunk Terstruktur

JSONL dipakai untuk menyimpan *chunk* yang sudah siap masuk pipeline RAG.

Contoh:

```
{"chunk_id":"DOC001-0001","document_id":"DOC001","title":"Panduan Akademik Mahasiswa","section":"Pengisian KRS","page_start":5,"page_end":6,"text":"Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai."}
```

Kelebihan JSONL:

- satu baris = satu *chunk*,
- mudah diproses baris per baris,
- cocok untuk corpus besar,
- metadata per *chunk* tetap ikut tersimpan,
- mudah divalidasi dengan Python.

JSON Lines secara resmi mensyaratkan tiga hal utama: encoding UTF-8, setiap baris merupakan JSON valid, dan setiap baris dipisahkan oleh karakter newline. (jsonlines.org)

Catatan kritis: JSONL bukan sekadar teks. Jika satu baris rusak, pipeline ingestion bisa gagal.

7.7 CSV untuk Metadata dan Evaluasi

CSV dipakai untuk menyimpan metadata ringkas.

Contoh `metadata.csv`:

```
document_id,title,source_file,clean_file,chunk_file,language,document_type,status,notes
DOC001,Panduan Akademik Mahasiswa,panduan-akademik.pdf,corpus_for_rag.md,corpus_all.jsonl,id,panduan,ready,Siap upload
```

Metadata membantu menjawab:

- dokumen apa yang diupload,
- sumber PDF-nya apa,
- hasil clean corpus ada di mana,
- hasil chunk ada di mana,
- statusnya siap atau belum,
- catatan kualitasnya apa.

Metadata adalah alat audit. Tanpa metadata, corpus sulit dipercaya.

7.8 Struktur Folder `rag_upload/`

Folder final:

```
data/rag_upload/  
├── corpus_for_rag.md  
├── corpus_all.jsonl  
├── metadata.csv  
└── upload_readme.md
```

Fungsi setiap file:

File	Fungsi
<code>corpus_for_rag.md</code>	Dokumen bersih final untuk upload RAG
<code>corpus_all.jsonl</code>	Semua chunk terstruktur
<code>metadata.csv</code>	Ringkasan identitas dan status corpus
<code>upload_readme.md</code>	Panduan isi folder dan cara pakai

Aturan: semua file di folder ini harus siap dibagikan ke sistem RAG. Jangan masukkan file mentah.

7.9 Praktik Python: Membuat Paket Upload RAG

Tujuan Praktik

Membaca:

```
data/clean/document_001.clean.md  
data/chunks/document_001.chunks.jsonl  
data/metadata/corpus_index.csv
```

Menghasilkan:

```
data/rag_upload/corpus_for_rag.md  
data/rag_upload/corpus_all.jsonl  
data/rag_upload/metadata.csv  
data/rag_upload/upload_readme.md
```

Script: `scripts/prepare_rag_upload.py`

```
from pathlib import Path
import shutil
import json
import pandas as pd

CLEAN_FILE = Path("data/clean/document_001.clean.md")
CHUNK_FILE = Path("data/chunks/document_001.chunks.jsonl")
CORPUS_INDEX = Path("data/metadata/corpus_index.csv")

UPLOAD_DIR = Path("data/rag_upload")
UPLOAD_MD = UPLOAD_DIR / "corpus_for_rag.md"
UPLOAD_JSONL = UPLOAD_DIR / "corpus_all.jsonl"
UPLOAD_METADATA = UPLOAD_DIR / "metadata.csv"
UPLOAD_README = UPLOAD_DIR / "upload_readme.md"

def validate_file_exists(file_path):
    if not file_path.exists():
        raise FileNotFoundError(f"File tidak ditemukan: {file_path}")

def validate_jsonl(jsonl_path):
    errors = []
    total_lines = 0

    with jsonl_path.open("r", encoding="utf-8") as file:
        for line_number, line in enumerate(file, start=1):
            line = line.strip()

            if not line:
                continue

            total_lines += 1

            try:
                data = json.loads(line)
            except json.JSONDecodeError as error:
                errors.append(f"Baris {line_number}: JSON tidak valid: {error}")
                continue

            required_fields = [
                "chunk_id",
                "document_id",
                "title",
                "section",
```

```

        "page_start",
        "page_end",
        "text",
    ]

    for field in required_fields:
        if field not in data:
            errors.append(f"Baris {line_number}:
field hilang: {field}")

        if "text" in data and len(data["text"].strip()) <
20:
            errors.append(f"Baris {line_number}: text
terlalu pendek")

    return total_lines, errors

def create_upload_readme(total_chunks, validation_errors):
    status = "READY" if not validation_errors else
"NEEDS_CHECK"

    error_text = "\n".join([f"- {error}" for error in
validation_errors])
    if not error_text:
        error_text = "- Tidak ada error JSONL terdeteksi."

    content = f"""# RAG Upload Package

## Status
{status}

## Isi Folder
- corpus_for_rag.md: corpus bersih dalam format Markdown.
- corpus_all.jsonl: chunk terstruktur untuk pipeline RAG.
- metadata.csv: metadata corpus.
- upload_readme.md: catatan paket upload.

## Jumlah Chunk
{total_chunks}

## Validasi JSONL
{error_text}

## Catatan
File dalam folder ini adalah file final yang siap diuji pada
Knowledge Base RAG.
"""

```

```

UPLOAD_README.write_text(content, encoding="utf-8")

def main():
    validate_file_exists(CLEAN_FILE)
    validate_file_exists(CHUNK_FILE)
    validate_file_exists(CORPUS_INDEX)

    UPLOAD_DIR.mkdir(parents=True, exist_ok=True)

    shutil.copyfile(CLEAN_FILE, UPLOAD_MD)
    shutil.copyfile(CHUNK_FILE, UPLOAD_JSONL)

    df = pd.read_csv(CORPUS_INDEX)
    df["rag_markdown_file"] = str(UPLOAD_MD)
    df["rag_jsonl_file"] = str(UPLOAD_JSONL)
    df["rag_status"] = "ready_for_validation"
    df.to_csv(UPLOAD_METADATA, index=False, encoding="utf-8")

    total_chunks, validation_errors =
validate_jsonl(UPLOAD_JSONL)
    create_upload_readme(total_chunks, validation_errors)

    print(f"File Markdown RAG dibuat: {UPLOAD_MD}")
    print(f"File JSONL RAG dibuat: {UPLOAD_JSONL}")
    print(f"Metadata upload dibuat: {UPLOAD_METADATA}")
    print(f"README upload dibuat: {UPLOAD_README}")

    if validation_errors:
        print("PERINGATAN: Ada error validasi. Cek
upload_readme.md")
    else:
        print("Validasi awal berhasil. Paket upload siap
diuji.")

if __name__ == "__main__":
    main()

```

7.10 Penjelasan Script

Bagian 1 — Input dan Output

```
CLEAN_FILE = Path("data/clean/document_001.clean.md")
CHUNK_FILE =
Path("data/chunks/document_001.chunks.jsonl")
CORPUS_INDEX = Path("data/metadata/corpus_index.csv")
```

Bagian ini menentukan file sumber dari Bab 5 dan Bab 6.

```
UPLOAD_DIR = Path("data/rag_upload")
```

Bagian ini menentukan folder final upload.

Bagian 2 — Validasi File Ada

```
def validate_file_exists(file_path):
    if not file_path.exists():
        raise FileNotFoundError(f"File tidak ditemukan:
{file_path}")
```

Fungsi ini memastikan file penting tidak hilang.

Tujuannya sederhana: jangan membuat paket upload dari file yang tidak lengkap.

Bagian 3 — Validasi JSONL

```
def validate_jsonl(jsonl_path):
```

Fungsi ini membaca file JSONL baris demi baris.

Yang dicek:

- apakah setiap baris JSON valid,
- apakah field wajib tersedia,
- apakah teks chunk terlalu pendek.

Field wajib:

```
chunk_id
document_id
title
section
page_start
page_end
text
```

Ini mengikuti struktur *chunk* yang sudah ditekankan dalam outline buku.

Bagian 4 — Membuat README Upload

```
def create_upload_readme(total_chunks,
                          validation_errors):
```

README ini penting agar orang lain tahu isi folder.

Isi README:

- status paket,
- isi folder,
- jumlah chunk,
- hasil validasi JSONL,
- catatan penggunaan.

README adalah bukti disiplin proyek.

Bagian 5 — Menyalin File Final

```
shutil.copyfile(CLEAN_FILE, UPLOAD_MD)
shutil.copyfile(CHUNK_FILE, UPLOAD_JSONL)
```

Bagian ini menyalin file clean corpus dan chunk ke folder `rag_upload/`.

Penting: file final upload tidak boleh diambil langsung dari folder eksperimen. Simpan salinan final agar proses lebih rapi.

Bagian 6 — Membuat Metadata Upload

```
df = pd.read_csv(CORPUS_INDEX)
df["rag_markdown_file"] = str(UPLOAD_MD)
df["rag_jsonl_file"] = str(UPLOAD_JSONL)
df["rag_status"] = "ready_for_validation"
```

Metadata diperluas agar tahu file mana yang siap upload.

7.11 Menjalankan Script

Jalankan:

```
python scripts/prepare_rag_upload.py
```

Output yang diharapkan:

```
data/rag_upload/corpus_for_rag.md
data/rag_upload/corpus_all.jsonl
data/rag_upload/metadata.csv
data/rag_upload/upload_readme.md
```

Contoh pesan berhasil:

```
File Markdown RAG dibuat: data/rag_upload/corpus_for_rag.md
File JSONL RAG dibuat: data/rag_upload/corpus_all.jsonl
Metadata upload dibuat: data/rag_upload/metadata.csv
README upload dibuat: data/rag_upload/upload_readme.md
Validasi awal berhasil. Paket upload siap diuji.
```

7.12 Contoh `corpus_for_rag.md`

```
---
document_id: "DOC001"
title: "Panduan Akademik Mahasiswa"
source_file: "panduan-akademik.pdf"
year: 2026
language: "id"
document_type: "panduan"
extraction_tool: "docling"
pages: 32
status: "ready_for_rag"
---

# Panduan Akademik Mahasiswa

## Pengisian KRS

Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.

## Persetujuan Dosen Wali

KRS harus disetujui oleh Dosen Wali.
```

Ciri file bagus:

- ada metadata,
- heading rapi,
- teks bersih,
- tidak ada log error,
- tidak ada catatan eksperimen,
- siap diuji.

7.13 Contoh `corpus_all.jsonl`

```
{ "chunk_id": "DOC001-0001", "document_id": "DOC001", "title": "Panduan Akademik Mahasiswa", "section": "Pengisian KRS", "page_start": 5, "page_end": 6, "text": "Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai." }
{ "chunk_id": "DOC001-0002", "document_id": "DOC001", "title": "Panduan Akademik Mahasiswa", "section": "Persetujuan Dosen Wali", "page_start": 6, "page_end": 6, "text": "KRS harus disetujui oleh Dosen Wali sebelum mahasiswa mengikuti perkuliahan." }
```

Ciri JSONL bagus:

- satu baris satu *chunk*,
- setiap baris JSON valid,
- semua field wajib ada,
- teks tidak kosong,
- metadata per chunk lengkap.

7.14 Contoh `metadata.csv`

```
document_id,title,source_file,clean_file,year,language,document_type,extraction_tool,pages,status,notes,rag_markdown_file,rag_jsonl_file,rag_status
DOC001,Panduan Akademik Mahasiswa,panduan-akademik.pdf,data/clean/document_001.clean.md,2026,id,panduan,docling,32,clean,Siap untuk chunking,data/rag_upload/corpus_for_rag.md,data/rag_upload/corpus_all.jsonl,ready_for_validation
```

Ciri metadata bagus:

- identitas dokumen jelas,
- sumber asli tercatat,
- lokasi file final jelas,
- status upload jelas,
- catatan kualitas tersedia.

7.15 Validasi Kesiapan Upload

Sebelum upload ke *Knowledge Base*, lakukan validasi.

Checklist Markdown

- Apakah file Markdown bisa dibaca manusia?
- Apakah heading rapi?
- Apakah metadata benar?
- Apakah teks bersih dari header-footer?
- Apakah tidak ada potongan error?

Checklist JSONL

- Apakah setiap baris JSON valid?
- Apakah setiap chunk punya **chunk_id**?
- Apakah setiap chunk punya **document_id**?
- Apakah setiap chunk punya **section**?
- Apakah teks chunk tidak kosong?
- Apakah chunk tidak terlalu pendek?

Checklist CSV

- Apakah **document_id** konsisten?
- Apakah nama file benar?
- Apakah status sudah **ready_for_validation**?
- Apakah lokasi file final benar?

Open WebUI Knowledge Base dapat dibuat dari menu Workspace > Knowledge, lalu file dapat diupload atau dokumen yang sudah ada dapat ditambahkan; Knowledge Base kemudian dapat dilampirkan ke model atau direferensikan dalam chat dengan #. ([Open WebUI](#))

7.16 Catatan Upload ke Knowledge Base

Pada tahap ini, mahasiswa belum fokus pada konfigurasi RAG secara mendalam. Fokus Bab 7 adalah **membuat file final yang siap upload**.

Alur umum:

```
data/rag_upload/  
→ corpus_for_rag.md  
→ upload ke Knowledge Base  
→ uji pertanyaan dasar
```

Untuk sistem yang menerima dokumen langsung, gunakan:

```
corpus_for_rag.md
```

Untuk pipeline yang dikendalikan dengan script, gunakan:

```
corpus_all.jsonl  
metadata.csv
```

Open WebUI juga mendokumentasikan bahwa dokumen lokal harus diupload melalui bagian Documents/Workspace agar dapat diakses dengan simbol # sebelum query. ([Open WebUI](#))

Prinsip penting: upload bukan akhir. Setelah upload, corpus harus diuji dengan pertanyaan.

7.17 Troubleshooting Ringkas

Masalah	Penyebab Umum	Solusi
File upload terlalu besar	Corpus belum dipilah	Pisahkan per dokumen atau per topik
Jawaban RAG tidak tepat	Chunk buruk atau metadata lemah	Kembali ke Bab 6
Banyak noise muncul	Cleaning belum cukup	Kembali ke Bab 5
JSONL gagal dibaca	Ada baris JSON rusak	Jalankan validasi JSONL
Metadata tidak cocok	<code>document_id</code> tidak konsisten	Samakan ID antar file
Markdown sulit dibaca	Heading dan paragraf rusak	Rapikan clean corpus
RAG mengambil konteks salah	Section terlalu umum	Perbaiki heading dan chunk

Jangan memperbaiki RAG hanya dari sisi model. Periksa dulu file upload.

7.18 Praktik Utama Bab 7

Tugas Praktik

Mahasiswa harus:

1. Membuat folder `data/rag_upload/`.
2. Menyalin clean corpus menjadi `corpus_for_rag.md`.
3. Menyalin chunk JSONL menjadi `corpus_all.jsonl`.
4. Membuat `metadata.csv`.
5. Membuat `upload_readme.md`.
6. Memvalidasi JSONL.
7. Memastikan paket siap upload ke *Knowledge Base*.

Output Wajib

```
data/rag_upload/  
├── corpus_for_rag.md  
├── corpus_all.jsonl  
├── metadata.csv  
└── upload_readme.md
```

Ringkasan Bab 7

Bab ini membedakan **file kerja** dan **file final upload**. Mahasiswa belajar menyiapkan paket corpus RAG yang terdiri dari Markdown bersih, JSONL chunk terstruktur, CSV metadata, dan README upload. File ini menjadi jembatan penting sebelum masuk ke tahap ekstraksi knowledge dan evaluasi RAG.

Kalimat kunci Bab 7:

RAG yang siap diuji dimulai dari folder upload yang bersih, konsisten, dan dapat diaudit.

BAB 8 — Ekstraksi Knowledge: Entitas, Konsep, Relasi, dan Evidence

Knowledge extraction yang baik tidak hanya menemukan istilah, tetapi selalu menyimpan bukti teks agar setiap relasi dapat diaudit.

8.1 Pembuka Bab — Dari Corpus ke Knowledge Layer

Bab 7 sudah menghasilkan paket upload RAG. Namun, corpus yang hanya berisi teks masih memiliki keterbatasan. Ia bisa dicari, tetapi belum tentu mudah dianalisis sebagai pengetahuan.

Bab 8 menambahkan **knowledge layer**: entitas, konsep, relasi, dan *evidence*. Tidak semua RAG membutuhkan ontologi formal. Tetapi corpus yang memiliki tabel entitas, konsep, relasi, dan bukti teks akan lebih mudah:

- diuji,
- diperbaiki,
- dijelaskan,
- dikembangkan menjadi ontologi ringan,
- dipakai untuk membuat pertanyaan evaluasi.

Outline buku menegaskan bahwa Bab 8 menghasilkan `knowledge/entities.csv`, `knowledge/concepts.csv`, `knowledge/relations.csv`, dan `knowledge/evidence_table.csv`.

Intinya: RAG yang kuat bukan hanya menyimpan teks, tetapi juga memahami struktur pengetahuan di balik teks.

8.2 Peta Halaman Bab 8

Bagian	Fokus	Estimasi
8.1–8.3	Konsep entitas, konsep, relasi, evidence	2 hlm
8.4–8.6	Keyword, entitas, konsep dari heading dan istilah	3 hlm
8.7–8.9	Relasi berbasis pola dan prompt LLM lokal	3 hlm
8.10–8.14	Praktik Python dan output CSV	3 hlm
8.15–8.17	Validasi, anti-halusinasi, ringkasan	1 hlm
Total		12 hlm

8.3 Apa Itu Entitas, Konsep, Istilah, Relasi, dan Evidence

Mahasiswa perlu membedakan lima istilah inti.

Istilah	Makna Praktis	Contoh
Entitas	Nama benda, aktor, unit, dokumen, sistem, atau objek spesifik	Mahasiswa, Dosen Wali, KRS
Konsep	Gagasan umum atau kategori yang muncul dalam dokumen	Registrasi Akademik, Pengisian KRS
Istilah	Kata penting yang sering muncul dan bermakna dalam domain	SKS, Semester, Cuti Akademik
Relasi	Hubungan antara dua entitas atau konsep	Mahasiswa mengisi KRS
Evidence	Bukti teks asli yang mendukung relasi	“Mahasiswa wajib mengisi KRS...”

Contoh:

Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.

Dapat diubah menjadi:

```
subject = Mahasiswa
relation = mengisi
object = KRS
evidence = Mahasiswa wajib mengisi KRS sebelum perkuliahan dimulai.
```

Relasi tanpa evidence tidak boleh dipercaya.

8.4 Mengapa Evidence Wajib Disimpan

Dalam RAG, *evidence* adalah pengaman dari halusinasi. Jika sebuah relasi tidak bisa ditelusuri ke kalimat sumber, relasi itu hanya dugaan.

Contoh relasi buruk:

subject	relation	object	evidence
Mahasiswa	membayar	KRS	kosong

Masalahnya:

- tidak jelas sumbernya,
- bisa salah,
- tidak bisa diaudit,
- tidak layak dipakai untuk ontologi,
- bisa memperburuk RAG.

Contoh relasi baik:

subject	relation	object	evidence
Mahasiswa	mengisi	KRS	"Mahasiswa wajib mengisi KRS..."

Dokumen outline menekankan bahwa setiap relasi harus memiliki bukti teks karena tanpa bukti, hasil ekstraksi bukan *knowledge extraction*, tetapi dugaan.

Aturan keras: tidak ada evidence, tidak masuk `relations.csv`.

8.5 Ekstraksi Keyword Sederhana dengan Python

Keyword adalah istilah penting yang sering muncul. Untuk tahap awal, gunakan metode sederhana sebelum memakai model berat.

Strategi sederhana:

- ambil kata atau frasa yang sering muncul,
- abaikan kata umum,
- ambil istilah dari heading,
- ambil kata berhuruf kapital,
- ambil istilah domain seperti KRS, SKS, KHS.

Contoh output:

```
term,count,source
KRS,18,DOC001
Mahasiswa,42,DOC001
Dosen Wali,9,DOC001
Mata Kuliah,14,DOC001
Semester,11,DOC001
```

Keyword bukan knowledge final. Ia hanya kandidat untuk diperiksa.

8.6 Ekstraksi Entitas dengan spaCy / Stanza

Untuk NLP, dua tool open source yang bisa dipakai adalah **spaCy** dan **Stanza**.

spaCy menyediakan *EntityRuler*, yaitu komponen untuk menambahkan entitas berdasarkan pola atau frasa; ini cocok untuk domain akademik karena istilah seperti KRS, SKS, Dosen Wali, dan Mata Kuliah bisa dibuat sebagai pola eksplisit. (spacy.io) Stanza adalah library NLP Python dari Stanford NLP Group yang mendukung analisis bahasa manusia, termasuk tokenisasi, lemma, POS, dependency parsing, dan pengenalan entitas untuk banyak bahasa. (stanfordnlp.github.io)

Untuk corpus Indonesia, pendekatan paling aman untuk mahasiswa:

- gunakan **rule-based extraction** untuk istilah domain,
- gunakan daftar istilah akademik,
- validasi manual,
- jangan terlalu percaya model bahasa umum.

Contoh daftar entitas:

```
DOMAIN_ENTITIES = [  
    "Mahasiswa",  
    "Dosen Wali",  
    "KRS",  
    "KHS",  
    "Mata Kuliah",  
    "SKS",  
    "Semester",  
    "Biro Akademik",  
]
```

Pesan kritis: model NLP membantu, tetapi domain knowledge tetap penting.

8.7 Ekstraksi Konsep dari Heading dan Istilah Berulang

Konsep sering muncul dari heading.

Contoh Markdown:

```
## Pengisian KRS
## Persetujuan Dosen Wali
## Cuti Akademik
## Evaluasi Studi
```

Dari heading ini, kandidat konsep:

```
Pengisian KRS
Persetujuan Dosen Wali
Cuti Akademik
Evaluasi Studi
```

Keet menempatkan *text processing to extract content for ontologies* dalam bagian *bottom-up ontology development*, sehingga cocok menjadi dasar praktis untuk mengubah teks corpus menjadi kandidat konsep dan relasi.

Konsep juga bisa berasal dari istilah berulang:

- KRS,
- KHS,
- SKS,
- Mahasiswa Aktif,
- Semester,
- Mata Kuliah,
- Dosen Wali.

Konsep yang baik biasanya penting, berulang, dan punya relasi dengan istilah lain.

8.8 Ekstraksi Relasi Berbasis Pola Kalimat

Relasi dapat diekstrak dengan pola sederhana.

Contoh pola:

X wajib mengisi Y
X disetujui oleh Y
X berisi Y
X memiliki Y
X mengajukan Y

Contoh hasil:

Kalimat	Relasi
Mahasiswa wajib mengisi KRS.	Mahasiswa — mengisi — KRS
KRS disetujui oleh Dosen Wali.	Dosen Wali — menyetujui — KRS
KRS berisi daftar Mata Kuliah.	KRS — berisi — Mata Kuliah
Mata Kuliah memiliki SKS.	Mata Kuliah — memiliki — SKS

Dalam Semantic Web, relasi sering direpresentasikan sebagai triple **subject–predicate–object**. Referensi terlampir juga menjelaskan RDF sebagai model triple yang fleksibel untuk merepresentasikan hubungan antar-entitas.

Pola sederhana lebih mudah diaudit daripada ekstraksi otomatis yang tidak jelas.

8.9 Ekstraksi Relasi Berbasis Prompt LLM Lokal

LLM lokal dapat membantu membuat kandidat relasi. Tetapi hasilnya harus diperlakukan sebagai **draft**, bukan kebenaran final.

Gunakan prompt yang memaksa model menyertakan evidence.

Contoh prompt:

```
Ekstrak relasi dari teks berikut.
```

```
Aturan:
```

1. Hanya gunakan informasi eksplisit dari teks.
2. Jangan menambah informasi dari luar teks.
3. Setiap relasi harus memiliki evidence berupa kutipan kalimat sumber.
4. Jika relasi tidak jelas, jangan keluarkan relasi.
5. Format output JSON array.

```
Teks:
```

```
[masukkan chunk di sini]
```

Output:

```
[
  {
    "subject": "",
    "relation": "",
    "object": "",
    "evidence": ""
  }
]
```

Ollama dapat dipakai untuk menjalankan model terbuka secara lokal; dokumentasi resminya menekankan penggunaan model terbuka dan instalasi lokal melalui terminal. ([Ollama](#))

Catatan kritis: LLM lokal bisa membantu mempercepat kerja, tetapi tetap bisa salah. Validasi manusia tetap wajib.

8.10 Mencegah Halusinasi Knowledge Extraction

Halusinasi dalam *knowledge extraction* terjadi ketika sistem membuat relasi yang tidak ada di teks.

Contoh teks sumber:

```
Mahasiswa wajib mengisi KRS.
```

Relasi valid:

```
Mahasiswa — mengisi — KRS
```

Relasi halusinatif:

```
Mahasiswa — membayar — KRS
```

Karena kata “membayar” tidak ada di teks.

Strategi anti-halusinasi:

- wajibkan evidence,
- batasi relasi pada teks eksplisit,
- simpan sumber dokumen,
- simpan section,
- simpan chunk_id,
- validasi manual,
- gunakan status *candidate*, *verified*, atau *rejected*.

Contoh kolom status:

```
subject,relation,object,evidence,source,chunk_id,status  
Mahasiswa,mengisi,KRS,"Mahasiswa wajib mengisi  
KRS...",DOC001,DOC001-0001,verified
```

Knowledge yang tidak bisa diaudit tidak boleh masuk tahap ontologi.

8.11 Praktik Python: Ekstraksi Entitas, Konsep, Relasi, dan Evidence

Tujuan Praktik

Membaca:

```
data/rag_upload/corpus_all.jsonl
```

Menghasilkan:

```
data/knowledge/entities.csv
data/knowledge/concepts.csv
data/knowledge/relations.csv
data/knowledge/evidence_table.csv
```

Script: scripts/extract_knowledge.py

```
from pathlib import Path
import json
import re
from collections import Counter
import pandas as pd

INPUT_JSONL = Path("data/rag_upload/corpus_all.jsonl")

ENTITIES_FILE = Path("data/knowledge/entities.csv")
CONCEPTS_FILE = Path("data/knowledge/concepts.csv")
RELATIONS_FILE = Path("data/knowledge/relations.csv")
EVIDENCE_FILE = Path("data/knowledge/evidence_table.csv")

DOMAIN_ENTITIES = [
    "Mahasiswa",
    "Dosen Wali",
    "KRS",
    "KHS",
    "Mata Kuliah",
    "SKS",
    "Semester",
    "Biro Akademik",
    "Cuti Akademik",
]

STOPWORDS = {
    "yang", "dan", "di", "ke", "dari", "untuk", "dengan",
    "atau", "pada", "dalam", "adalah", "sebagai", "oleh",
    "ini", "itu", "harus", "wajib", "dapat",
}
```

```

def load_chunks(jsonl_path):
    chunks = []

    with jsonl_path.open("r", encoding="utf-8") as file:
        for line in file:
            line = line.strip()

            if not line:
                continue

            chunks.append(json.loads(line))

    return chunks

def split_sentences(text):
    sentences = re.split(r"(?<=[.!?])\s+", text)
    return [sentence.strip() for sentence in sentences if
sentence.strip()]

def extract_entities(chunks):
    rows = []

    for chunk in chunks:
        text = chunk["text"]

        for entity in DOMAIN_ENTITIES:
            if re.search(rf"\b{re.escape(entity)}\b", text,
flags=re.IGNORECASE):
                rows.append({
                    "entity": entity,
                    "entity_type": "DOMAIN_TERM",
                    "source": chunk["document_id"],
                    "chunk_id": chunk["chunk_id"],
                    "section": chunk["section"],
                })

    return rows

def extract_concepts(chunks):
    concept_counter = Counter()

    for chunk in chunks:
        section = chunk.get("section", "").strip()

```

```

        if section and section.lower() != "tanpa section":
            concept_counter[section] += 1

        words = re.findall(r"\b[A-Za-zÀ-ÿ]{4,}\b",
chunk["text"])

        for word in words:
            word_lower = word.lower()

            if word_lower not in STOPWORDS:
                concept_counter[word] += 1

rows = []

for concept, count in concept_counter.most_common(50):
    rows.append({
        "concept": concept,
        "frequency": count,
        "status": "candidate",
    })

return rows

def find_sentence_with_terms(sentences, terms):
    for sentence in sentences:
        found_all = True

        for term in terms:
            if not re.search(rf"\b{re.escape(term)}\b",
sentence, flags=re.IGNORECASE):
                found_all = False
                break

        if found_all:
            return sentence

    return ""

def extract_relations(chunks):
    relation_patterns = [
        {
            "subject": "Mahasiswa",
            "relation": "mengisi",
            "object": "KRS",
            "terms": ["Mahasiswa", "KRS"],

```

```

        "verbs": ["mengisi", "wajib mengisi"],
    },
    {
        "subject": "Dosen Wali",
        "relation": "menyetujui",
        "object": "KRS",
        "terms": ["Dosen Wali", "KRS"],
        "verbs": ["menyetujui", "disetujui"],
    },
    {
        "subject": "KRS",
        "relation": "berisi",
        "object": "Mata Kuliah",
        "terms": ["KRS", "Mata Kuliah"],
        "verbs": ["berisi", "memuat"],
    },
    {
        "subject": "Mata Kuliah",
        "relation": "memiliki",
        "object": "SKS",
        "terms": ["Mata Kuliah", "SKS"],
        "verbs": ["memiliki", "berjumlah"],
    },
]

rows = []

for chunk in chunks:
    sentences = split_sentences(chunk["text"])

    for pattern in relation_patterns:
        evidence = find_sentence_with_terms(sentences,
pattern["terms"])

        if not evidence:
            continue

        has_verb = any(
            verb.lower() in evidence.lower()
            for verb in pattern["verbs"]
        )

        if not has_verb:
            continue

        rows.append({
            "subject": pattern["subject"],
            "relation": pattern["relation"],

```

```

        "object": pattern["object"],
        "evidence": evidence,
        "source": chunk["document_id"],
        "chunk_id": chunk["chunk_id"],
        "section": chunk["section"],
        "status": "candidate",
    })

    return rows

def create_evidence_table(relations):
    rows = []

    for index, relation in enumerate(relations, start=1):
        rows.append({
            "evidence_id": f"EV{index:04d}",
            "subject": relation["subject"],
            "relation": relation["relation"],
            "object": relation["object"],
            "evidence": relation["evidence"],
            "source": relation["source"],
            "chunk_id": relation["chunk_id"],
            "section": relation["section"],
        })

    return rows

def save_csv(rows, output_file):
    output_file.parent.mkdir(parents=True, exist_ok=True)
    df = pd.DataFrame(rows)
    df.to_csv(output_file, index=False, encoding="utf-8")

def main():
    if not INPUT_JSONL.exists():
        raise FileNotFoundError(f"File tidak ditemukan: {INPUT_JSONL}")

    chunks = load_chunks(INPUT_JSONL)

    entities = extract_entities(chunks)
    concepts = extract_concepts(chunks)
    relations = extract_relations(chunks)
    evidence_table = create_evidence_table(relations)

    save_csv(entities, ENTITIES_FILE)

```

```
save_csv(concepts, CONCEPTS_FILE)
save_csv(relations, RELATIONS_FILE)
save_csv(evidence_table, EVIDENCE_FILE)

print(f"Entities dibuat: {ENTITIES_FILE}")
print(f"Concepts dibuat: {CONCEPTS_FILE}")
print(f"Relations dibuat: {RELATIONS_FILE}")
print(f"Evidence table dibuat: {EVIDENCE_FILE}")
print(f"Jumlah entitas: {len(entities)}")
print(f"Jumlah konsep: {len(concepts)}")
print(f"Jumlah relasi: {len(relations)}")

if __name__ == "__main__":
    main()
```

8.12 Penjelasan Script

Input

```
INPUT_JSONL = Path("data/rag_upload/corpus_all.jsonl")
```

Script membaca *chunk* final dari Bab 7. Ini penting karena knowledge harus diambil dari file final, bukan file mentah.

Domain Entities

```
DOMAIN_ENTITIES = [  
    "Mahasiswa",  
    "Dosen Wali",  
    "KRS",  
    "KHS",  
    "Mata Kuliah",  
    "SKS",  
    "Semester",  
    "Biro Akademik",  
    "Cuti Akademik",  
]
```

Daftar ini adalah kamus istilah domain. Untuk kasus lain, mahasiswa dapat mengganti daftar ini.

Contoh:

- SOP laboratorium: Laboran, Mahasiswa, Alat, Bahan, Form Peminjaman.
- Layanan kampus: Pemohon, Petugas, Dokumen, Layanan.

Ekstraksi Entitas

```
if re.search(rf"\b{re.escape(entity)}\b", text,  
            flags=re.IGNORECASE):
```

Baris ini mencari entitas secara aman di dalam teks.

- `re.escape()` mencegah karakter khusus merusak pola.
- `IGNORECASE` membuat pencarian tidak sensitif huruf besar/kecil.

Ekstraksi Konsep

```
concept_counter[section] += 1
```

Heading section dianggap sebagai kandidat konsep.

```
concept_counter[word] += 1
```

Kata yang sering muncul juga dianggap kandidat konsep.

Catatan: hasil konsep masih kandidat. Harus divalidasi.

Ekstraksi Relasi

```
relation_patterns = [...]
```

Relasi dibuat dari pola domain.

Contoh:

```
{
  "subject": "Mahasiswa",
  "relation": "mengisi",
  "object": "KRS",
  "terms": ["Mahasiswa", "KRS"],
  "verbs": ["mengisi", "wajib mengisi"],
}
```

Relasi hanya dibuat jika:

- subject dan object muncul dalam satu kalimat,
- kata kerja relasi juga muncul,
- evidence tersedia.

Ini lebih aman daripada menebak relasi.

Evidence Table

```
"evidence_id": f"EV{index:04d}"
```

Setiap evidence diberi ID agar bisa dirujuk saat validasi.

8.13 Menjalankan Script

Jalankan:

```
python scripts/extract_knowledge.py
```

Output:

```
data/knowledge/entities.csv  
data/knowledge/concepts.csv  
data/knowledge/relations.csv  
data/knowledge/evidence_table.csv
```

Contoh pesan:

```
Entities dibuat: data/knowledge/entities.csv  
Concepts dibuat: data/knowledge/concepts.csv  
Relations dibuat: data/knowledge/relations.csv  
Evidence table dibuat: data/knowledge/evidence_table.csv  
Jumlah entitas: 24  
Jumlah konsep: 50  
Jumlah relasi: 3
```

8.14 Contoh Output `entities.csv`

```
entity,entity_type,source,chunk_id,section
Mahasiswa,DOMAIN_TERM,DOC001,DOC001-0001,Pengisian KRS
KRS,DOMAIN_TERM,DOC001,DOC001-0001,Pengisian KRS
Dosen Wali,DOMAIN_TERM,DOC001,DOC001-0002,Persetujuan Dosen
Wali
```

Ciri output bagus:

- entitas jelas,
- sumber tercatat,
- chunk tercatat,
- section tercatat.

8.15 Contoh Output `concepts.csv`

```
concept,frequency,status
Pengisian KRS,3,candidate
Persetujuan Dosen Wali,2,candidate
Mahasiswa,42,candidate
Semester,11,candidate
Mata Kuliah,14,candidate
```

Status awal `candidate` penting karena konsep belum tentu benar.

Rekomendasi status:

Status	Arti
<code>candidate</code>	Hasil ekstraksi awal
<code>verified</code>	Sudah dicek manusia
<code>rejected</code>	Tidak layak dipakai
<code>merged</code>	Digabung dengan konsep lain

8.16 Contoh Output `relations.csv`

```
subject,relation,object,evidence,source,chunk_id,section,status
Mahasiswa,mengisi,KRS,"Mahasiswa wajib mengisi KRS sebelum
perkuliahan dimulai.",DOC001,DOC001-0001,Pengisian
KRS,candidate
Dosen Wali,menyetujui,KRS,"KRS disetujui oleh Dosen
Wali.",DOC001,DOC001-0002,Persetujuan Dosen Wali,candidate
KRS,berisi,Mata Kuliah,"KRS berisi daftar Mata Kuliah yang
diambil mahasiswa.",DOC001,DOC001-0003,Isi KRS,candidate
```

Inilah format yang akan menjadi dasar Bab 9.

8.17 Contoh Output `evidence_table.csv`

```
evidence_id,subject,relation,object,evidence,source,chunk_id,
section
EV0001,Mahasiswa,mengisi,KRS,"Mahasiswa wajib mengisi KRS
sebelum perkuliahan dimulai.",DOC001,DOC001-0001,Pengisian
KRS
EV0002,Dosen Wali,menyetujui,KRS,"KRS disetujui oleh Dosen
Wali.",DOC001,DOC001-0002,Persetujuan Dosen Wali
EV0003,KRS,berisi,Mata Kuliah,"KRS berisi daftar Mata Kuliah
yang diambil mahasiswa.",DOC001,DOC001-0003,Isi KRS
```

Evidence table adalah jembatan antara teks dan knowledge graph.

8.18 Validasi Manual

Setelah file CSV dibuat, mahasiswa harus memeriksa hasilnya.

Checklist validasi:

- Apakah entitas benar-benar muncul di teks?
- Apakah konsep bukan kata umum?
- Apakah relasi didukung evidence?
- Apakah evidence berasal dari chunk yang benar?
- Apakah subject dan object tidak tertukar?
- Apakah relasi terlalu umum?
- Apakah ada relasi halusinatif?
- Apakah ada sinonim yang perlu digabung?

Contoh masalah:

Masalah	Contoh	Perbaikan
Subject-object tertukar	KRS menyetujui Dosen Wali	Dosen Wali menyetujui KRS
Relasi terlalu umum	Mahasiswa terkait KRS	Mahasiswa mengisi KRS
Evidence lemah	hanya kata “KRS”	cari kalimat lengkap
Duplikasi istilah	Dosen wali vs Dosen Wali	normalisasi nama entitas

Validasi manual adalah bagian dari metode, bukan pekerjaan tambahan.

8.19 Menghubungkan ke Ontologi Ringan

Bab 8 belum membuat ontologi final. Bab ini menyiapkan bahan untuk Bab 9.

Alurnya:

```
entities.csv
concepts.csv
relations.csv
evidence_table.csv
→ ontology/corpus_ontology.ttl
```

Referensi terlampir menjelaskan bahwa RDF, OWL, SPARQL, URI, dan reasoning menjadi bagian penting dari *semantic knowledge management*, termasuk ketika pengetahuan perlu direpresentasikan sebagai triple dan di-query.

Namun, untuk mahasiswa, targetnya tetap sederhana:

- buat entitas,
- buat relasi,
- simpan evidence,
- validasi,
- baru ubah ke ontologi ringan.

Jangan membuat ontologi dari relasi yang belum divalidasi.

8.20 Praktik Utama Bab 8

Tugas Praktik

Mahasiswa harus:

1. Membaca file `corpus_all.jsonl`.
2. Mengekstrak entitas dari daftar domain.
3. Mengekstrak konsep dari heading dan istilah berulang.
4. Mengekstrak relasi berbasis pola kalimat.
5. Menyimpan evidence untuk setiap relasi.
6. Membuat empat file CSV.
7. Memvalidasi hasil secara manual.

Output Wajib

```
data/knowledge/entities.csv
data/knowledge/concepts.csv
data/knowledge/relations.csv
data/knowledge/evidence_table.csv
```

8.21 Troubleshooting Ringkas

Masalah	Penyebab Umum	Solusi
Entitas tidak ditemukan	Daftar domain belum lengkap	Tambah <code>DOMAIN_ENTITIES</code>
Terlalu banyak konsep	Stopword kurang kuat	Tambah daftar stopwords
Relasi terlalu sedikit	Pola relasi terlalu sempit	Tambah pola kalimat
Relasi salah	Subject-object tertukar	Validasi manual
Evidence kosong	Kalimat tidak ditemukan	Jangan simpan relasi
Banyak duplikasi	Variasi penulisan istilah	Normalisasi istilah
LLM lokal mengarang	Prompt terlalu bebas	Paksa evidence dan format JSON
CSV sulit dibaca	Encoding tidak benar	Gunakan UTF-8

Ringkasan Bab 8

Bab ini mengubah corpus dari sekadar teks menjadi **knowledge layer**. Mahasiswa belajar mengekstrak entitas, konsep, relasi, dan evidence. Pendekatannya praktis: mulai dari keyword, heading, istilah domain, pola kalimat, lalu bantuan LLM lokal secara hati-hati. Semua relasi harus punya evidence agar tidak berubah menjadi halusinasi.

Kalimat kunci Bab 8:

Knowledge yang benar bukan yang terdengar masuk akal, tetapi yang dapat ditunjukkan kembali ke kalimat sumber.

BAB 9 — Ontologi Ringan untuk Memperkaya Corpus RAG

Ontologi ringan bukan untuk membuat mahasiswa tenggelam dalam logika formal; ontologi ringan dipakai agar corpus RAG punya peta konsep, relasi, dan konteks yang bisa diuji.

9.1 Ontologi sebagai Peta Pengetahuan

Bab 8 sudah menghasilkan `entities.csv`, `concepts.csv`, `relations.csv`, dan `evidence_table.csv`. Bab 9 mengubah hasil tersebut menjadi **peta pengetahuan ringan**.

Tujuannya bukan membuat ontologi akademik yang rumit. Tujuannya adalah membuat mahasiswa paham bahwa relasi dalam dokumen dapat diubah menjadi bentuk yang lebih terstruktur:

```
Mahasiswa → mengisi → KRS
Dosen Wali → menyetujui → KRS
KRS → berisi → Mata Kuliah
Mata Kuliah → memiliki → SKS
```

Outline buku menegaskan bahwa Bab 9 diarahkan untuk membuat **ontologi ringan**, mengubah `relations.csv` menjadi RDF triple, membuat file Turtle dengan `rdflib`, membuat diagram Mermaid, dan memakai ontologi sebagai metadata pencarian RAG.

Pesan utama: ontologi ringan adalah jembatan antara teks, *knowledge graph*, dan RAG.

9.2 Peta Halaman Bab 9

Bagian	Fokus	Estimasi
9.1–9.3	Ontologi ringan, taxonomy, knowledge graph	2 hlm
9.4–9.6	Class, instance, property, relation	2 hlm
9.7–9.10	RDF triple, Turtle, rdflib	3 hlm
9.11–9.13	Mermaid, Protégé, metadata RAG	2 hlm
9.14–9.16	Validasi, troubleshooting, ringkasan	1 hlm
Total		10 hlm

9.3 Ontologi Ringan untuk RAG, Bukan Ontologi Akademik yang Rumit

Untuk buku ini, **ontologi ringan** berarti struktur sederhana yang menjelaskan:

- konsep penting,
- aktor,
- dokumen,
- proses,
- relasi,
- sumber evidence.

Contoh:

```
Mahasiswa adalah aktor.  
KRS adalah dokumen akademik.  
Mahasiswa mengisi KRS.  
Dosen Wali menyetujui KRS.
```

Ontologi ringan berbeda dari ontologi akademik yang sangat formal. Keet membedakan kebutuhan ontologi ringan untuk anotasi, NLP, dan akses data dengan ontologi ilmiah yang lebih ekspresif dan membutuhkan reasoning lebih kuat. Untuk buku mahasiswa AI, pendekatan ringan lebih tepat karena fokusnya adalah **corpus RAG yang dapat dicari dan diaudit**, bukan kuliah logika formal mendalam.

Prinsip praktis: gunakan struktur yang cukup untuk membantu RAG, jangan membuat struktur yang terlalu berat untuk mahasiswa.

9.4 Taxonomy vs Knowledge Graph vs Ontology

Mahasiswa sering mencampur tiga istilah ini. Bedakan secara sederhana.

Istilah	Makna Praktis	Contoh
Taxonomy	Susunan hierarki sederhana	Dokumen Akademik → KRS → KHS
Knowledge Graph	Jaringan entitas dan relasi	Mahasiswa → mengisi → KRS
Ontologi	Struktur konsep, kelas, relasi, properti, dan makna	Mahasiswa adalah Aktor; KRS adalah Dokumen; mengisi adalah relasi

Contoh *taxonomy*:

```
Dokumen Akademik
├── KRS
├── KHS
└── Transkrip
```

Contoh *knowledge graph*:

```
Mahasiswa → mengisi → KRS
KRS → berisi → Mata Kuliah
Mata Kuliah → memiliki → SKS
```

Contoh ontologi ringan:

```
Class: Aktor
Class: DokumenAkademik
Class: MataKuliah

Instance: Mahasiswa
Instance: DosenWali
Instance: KRS

Property: mengisi
Property: menyetujui
Property: berisi
Property: memiliki
```

SKOS dapat dipakai untuk struktur istilah dan konsep ringan, sedangkan OWL dipakai jika perlu struktur ontologi yang lebih kaya. Keet juga menempatkan SKOS sebagai salah satu pilihan untuk anotasi dan struktur ringan, sementara OWL dipakai untuk kebutuhan yang lebih ekspresif.

9.5 Class, Instance, Property, dan Relation

Empat istilah ini cukup untuk Bab 9.

Class

Class adalah kategori umum.

Contoh:

```
Aktor
DokumenAkademik
MataKuliah
ProsesAkademik
```

Instance

Instance adalah contoh konkret dari *class*.

Contoh:

```
Mahasiswa adalah instance dari Aktor.
Dosen Wali adalah instance dari Aktor.
KRS adalah instance dari DokumenAkademik.
```

Property / Relation

Property atau relasi menjelaskan hubungan.

Contoh:

```
Mahasiswa mengisi KRS.
Dosen Wali menyetujui KRS.
KRS berisi Mata Kuliah.
```

OWL 2 menyebut objek sebagai *individuals*, kategori sebagai *classes*, dan relasi sebagai *properties*. OWL juga membedakan *object properties*, *datatype properties*, dan *annotation properties*. ([W3C](#))

Untuk mahasiswa: cukup mulai dari class, instance, dan relation. Jangan masuk terlalu dalam ke aksioma yang rumit.

9.6 RDF Triple: Subject–Predicate–Object

RDF menyimpan pengetahuan sebagai triple:

`subject → predicate → object`

Contoh:

`Mahasiswa → mengisi → KRS`
`DosenWali → menyetujui → KRS`
`KRS → berisi → MataKuliah`

W3C menjelaskan bahwa RDF graph adalah kumpulan triple **subject–predicate–object**, dan triple dipakai untuk menyatakan deskripsi tentang resource. ([W3C](#)) Dalam RDF Primer, predicate menjelaskan sifat hubungan antara subject dan object. ([W3C](#))

Dalam konteks Bab 9, `relations.csv` dari Bab 8 adalah bahan utama untuk membuat RDF triple.

Contoh `relations.csv`:

```
subject,relation,object,evidence,source,chunk_id,section,status
Mahasiswa,mengisi,KRS,"Mahasiswa wajib mengisi
KRS...",DOC001,DOC001-0001,Pengisian KRS,verified
Dosen Wali,menyetujui,KRS,"KRS disetujui oleh Dosen
Wali.",DOC001,DOC001-0002,Persetujuan Dosen Wali,verified
KRS,berisi,Mata Kuliah,"KRS berisi daftar Mata
Kuliah.",DOC001,DOC001-0003,Isi KRS,verified
```

9.7 Turtle: Format Ringkas untuk RDF

Turtle adalah salah satu format penulisan RDF yang lebih mudah dibaca manusia.

Contoh sederhana:

```
@prefix ex: <http://example.org/corpus#> .  
  
ex:Mahasiswa ex:mengisi ex:KRS .  
ex:DosenWali ex:menyetujui ex:KRS .  
ex:KRS ex:berisi ex:MataKuliah .
```

Kelebihan Turtle:

- lebih ringkas daripada RDF/XML,
- mudah dibaca,
- mudah dibuat dengan Python,
- cocok untuk latihan mahasiswa,
- bisa dibuka dengan tool RDF/OWL.

RDFLib mendukung serialisasi RDF ke banyak format, termasuk Turtle; dokumentasinya menunjukkan `Graph.serialize()` dapat menyimpan graph ke file `.ttl`.

(rdflib.readthedocs.io)

9.8 Praktik Python: Mengubah `relations.csv` menjadi Turtle

Tujuan Praktik

Membaca:

```
data/knowledge/relations.csv
```

Menghasilkan:

```
data/ontology/corpus_ontology.ttl  
data/ontology/corpus_ontology.owl
```

Script: `scripts/build_lightweight_ontology.py`

```
from pathlib import Path  
import re  
import pandas as pd  
from rdflib import Graph, Namespace, URIRef, Literal  
from rdflib.namespace import RDF, RDFS, OWL, XSD  
  
RELATIONS_FILE = Path("data/knowledge/relations.csv")  
OUTPUT_TTL = Path("data/ontology/corpus_ontology.ttl")  
OUTPUT_OWL = Path("data/ontology/corpus_ontology.owl")  
  
EX = Namespace("http://example.org/corpus#")  
  
def slugify(text):  
    text = str(text).strip()  
    text = text.replace(" ", "_")  
    text = re.sub(r"^[A-Za-z0-9_]", "", text)  
    return text  
  
def add_class(g, class_name):  
    class_uri = EX[slugify(class_name)]  
    g.add((class_uri, RDF.type, OWL.Class))  
    g.add((class_uri, RDFS.label, Literal(class_name,  
lang="id")))  
    return class_uri  
  
def add_individual(g, name, class_uri=None):  
    individual_uri = EX[slugify(name)]  
    g.add((individual_uri, RDF.type, OWL.NamedIndividual))
```

```

        g.add((individual_uri, RDFS.label, Literal(name,
lang="id")))

    if class_uri:
        g.add((individual_uri, RDF.type, class_uri))

    return individual_uri

def add_object_property(g, relation_name):
    property_uri = EX[slugify(relation_name)]
    g.add((property_uri, RDF.type, OWL.ObjectProperty))
    g.add((property_uri, RDFS.label, Literal(relation_name,
lang="id")))
    return property_uri

def build_graph(relations_df):
    g = Graph()
    g.bind("ex", EX)
    g.bind("rdf", RDF)
    g.bind("rdfs", RDFS)
    g.bind("owl", OWL)
    g.bind("xsd", XSD)

    actor_class = add_class(g, "Aktor")
    document_class = add_class(g, "Dokumen Akademik")
    course_class = add_class(g, "Mata Kuliah")
    concept_class = add_class(g, "Konsep")

    class_map = {
        "Mahasiswa": actor_class,
        "Dosen Wali": actor_class,
        "KRS": document_class,
        "KHS": document_class,
        "Mata Kuliah": course_class,
        "SKS": concept_class,
    }

    for _, row in relations_df.iterrows():
        subject = str(row["subject"]).strip()
        relation = str(row["relation"]).strip()
        obj = str(row["object"]).strip()

        if not subject or not relation or not obj:
            continue

```

```

        subject_uri = add_individual(g, subject,
class_map.get(subject, concept_class))
        object_uri = add_individual(g, obj,
class_map.get(obj, concept_class))
        property_uri = add_object_property(g, relation)

        g.add((subject_uri, property_uri, object_uri))

        if "evidence" in row and pd.notna(row["evidence"]):
            evidence_literal = Literal(str(row["evidence"]),
lang="id")
            g.add((subject_uri, EX.hasEvidence,
evidence_literal))

            if "source" in row and pd.notna(row["source"]):
                g.add((subject_uri, EX.hasSource,
Literal(str(row["source"]))))

            if "chunk_id" in row and pd.notna(row["chunk_id"]):
                g.add((subject_uri, EX.hasChunkId,
Literal(str(row["chunk_id"]))))

        return g

def main():
    if not RELATIONS_FILE.exists():
        raise FileNotFoundError(f"File tidak ditemukan:
{RELATIONS_FILE}")

    OUTPUT_TTL.parent.mkdir(parents=True, exist_ok=True)

    relations_df = pd.read_csv(RELATIONS_FILE)
    graph = build_graph(relations_df)

    graph.serialize(destination=OUTPUT_TTL, format="turtle")
    graph.serialize(destination=OUTPUT_OWL, format="xml")

    print(f"Ontology Turtle dibuat: {OUTPUT_TTL}")
    print(f"Ontology OWL/RDF XML dibuat: {OUTPUT_OWL}")
    print(f"Jumlah triple: {len(graph)}")

if __name__ == "__main__":
    main()

```

9.9 Penjelasan Script

Library Utama

```
from rdflib import Graph, Namespace, URIRef, Literal
from rdflib.namespace import RDF, RDFS, OWL, XSD
```

Penjelasan:

- **Graph** adalah wadah triple RDF.
- **Namespace** membuat prefix URI.
- **Literal** menyimpan teks biasa, seperti label dan evidence.
- **RDF, RDFS, OWL, XSD** adalah namespace standar.

RDFLib adalah library Python untuk bekerja dengan RDF; dokumentasinya menyebut bahwa RDFLib mendukung parser dan serializer untuk RDF/XML, Turtle, N-Triples, JSON-LD, dan juga mendukung SPARQL 1.1. ([GitHub](#))

Fungsi **slugify()**

```
def slugify(text):
```

Fungsi ini mengubah teks menjadi nama URI yang aman.

Contoh:

```
Dosen Wali → Dosen_Wali
Mata Kuliah → Mata_Kuliah
```

Membuat Class

```
g.add((class_uri, RDF.type, OWL.Class))
```

Baris ini menyatakan bahwa sebuah URI adalah *class* OWL.

Membuat Individual

```
g.add((individual_uri, RDF.type, OWL.NamedIndividual))
```

Baris ini menyatakan bahwa entitas adalah *individual*.

Membuat Object Property

```
g.add((property_uri, RDF.type, OWL.ObjectProperty))
```

Relasi seperti **mengisi**, **menyetujui**, dan **berisi** diperlakukan sebagai *object property*.

Menambahkan Evidence

```
g.add((subject_uri, EX.hasEvidence, evidence_literal))
```

Evidence tetap disimpan agar relasi tidak lepas dari sumber teks.

Catatan kritis: evidence tidak menggantikan validasi, tetapi membantu audit.

9.10 Contoh Output Turtle

```
@prefix ex: <http://example.org/corpus#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ex:Aktor a owl:Class ;
    rdfs:label "Aktor"@id .

ex:Dokumen_Akademik a owl:Class ;
    rdfs:label "Dokumen Akademik"@id .

ex:Mahasiswa a owl:NamedIndividual,
    ex:Aktor ;
    rdfs:label "Mahasiswa"@id ;
    ex:mengisi ex:KRS ;
    ex:hasEvidence "Mahasiswa wajib mengisi KRS..."@id ;
    ex:hasSource "DOC001" .

ex:Dosen_Wali a owl:NamedIndividual,
    ex:Aktor ;
    rdfs:label "Dosen Wali"@id ;
    ex:menyetujui ex:KRS .
```

Ciri output bagus:

- ada prefix,
- ada *class*,
- ada *individual*,
- ada relasi,
- ada label manusiawi,
- ada evidence dan sumber.

9.11 Membuat Diagram Mermaid dari Relasi

Diagram membantu mahasiswa melihat **peta pengetahuan** secara visual. Relasi yang sebelumnya tersimpan dalam bentuk tabel dapat diubah menjadi diagram agar hubungan antar-entitas lebih mudah dipahami.

Pada tahap ini, file `relations.csv` akan dibaca oleh Python, lalu dikonversi menjadi diagram berbasis *Mermaid*.

Output:

```
data/ontology/ontology_diagram.md
```

Script:

```
scripts/create_ontology_diagram.py
```

Isi script:

```
from pathlib import Path
import pandas as pd
import re

RELATIONS_FILE = Path("data/knowledge/relations.csv")
OUTPUT_FILE = Path("data/ontology/ontology_diagram.md")

def clean_node(text):
    text = str(text).strip()
    text = re.sub(r"^[A-Za-z0-9_À-ÿ ]", "", text)
    return text.replace(" ", "_")

def main():
    if not RELATIONS_FILE.exists():
        raise FileNotFoundError(f"File tidak ditemukan: {RELATIONS_FILE}")

    df = pd.read_csv(RELATIONS_FILE)

    lines = []
    lines.append("# Ontology Diagram\n")
    lines.append("`mermaid")
    lines.append("graph TD")

    for _, row in df.iterrows():
        subject = clean_node(row["subject"])
        relation = str(row["relation"]).strip()
        obj = clean_node(row["object"])
```

```

        lines.append(f"        {subject} -- {relation} -->
{obj}")

    lines.append("```")

    OUTPUT_FILE.parent.mkdir(parents=True, exist_ok=True)
    OUTPUT_FILE.write_text("\n".join(lines),
encoding="utf-8")

    print(f"Diagram ontologi dibuat: {OUTPUT_FILE}")

if __name__ == "__main__":
    main()

```

Jalankan perintah berikut:

```
python scripts/create_ontology_diagram.py
```

Jika berhasil, file berikut akan dibuat:

```
data/ontology/ontology_diagram.md
```

Contoh output:

```

# Ontology Diagram

```mermaid
graph TD
 Mahasiswa -- mengisi --> KRS
 Dosen_Wali -- menyetujui --> KRS
 KRS -- berisi --> Mata_Kuliah
 Mata_Kuliah -- memiliki --> SKS

```

Diagram di atas menunjukkan bahwa pengetahuan dalam corpus tidak hanya berupa paragraf, tetapi juga dapat dibaca sebagai **hubungan antar-konsep** Misalnya, **Mahasiswa** berelasi dengan **KRS**, **Dosen Wali** berelasi dengan **KRS**, dan **Mata Kuliah** berelasi dengan **SKS**.

Mermaid mendukung diagram berbasis teks seperti *flowchart* dan *graph*. Karena itu, Mermaid cocok untuk dokumentasi teknis yang sederhana, mudah dibaca, mudah disimpan di dalam repository, dan dapat dikelola bersama file corpus lainnya.

Catatan penting: diagram Mermaid bukan pengganti ontologi formal seperti RDF atau OWL. Diagram ini berfungsi sebagai **alat bantu visual** agar mahasiswa cepat memahami relasi utama sebelum masuk ke validasi ontologi yang lebih serius.



## 9.12 Membuka Hasil di Protégé

Protégé bersifat **opsional**, tetapi sangat berguna untuk melihat struktur ontologi secara lebih visual. Protégé adalah *ontology editor* berbasis *open source* yang mendukung format **OWL** dan **RDF**, serta banyak digunakan dalam praktik *ontology engineering*.

Tujuan membuka hasil di Protégé adalah untuk memastikan bahwa ontologi yang dibuat dari corpus benar-benar dapat dibaca, memiliki struktur yang rapi, dan tidak menghasilkan relasi atau nama entitas yang membingungkan.

Langkah sederhana:

1. Buka aplikasi **Protégé**.
2. Pilih menu **Open**.
3. Buka file berikut:

```
data/ontology/corpus_ontology.owl
```

Setelah file terbuka, cek beberapa bagian utama berikut:

- **Classes**
- **Object Properties**
- **Individuals**
- **Annotations**

Hal-hal penting yang perlu diperiksa:

- apakah **class** muncul dengan benar;
- apakah **individual** muncul;
- apakah **relasi** antar-entitas terbaca;
- apakah **label** dapat dibaca dengan jelas;
- apakah tidak ada nama URI yang aneh atau terlalu sulit dipahami;
- apakah **evidence** tersimpan sebagai *annotation* atau *property*.

Pemeriksaan ini penting karena ontologi yang terlihat benar secara teknis belum tentu mudah dipahami oleh manusia. Untuk kebutuhan RAG, ontologi ringan sebaiknya tetap sederhana, bersih, dan dapat ditelusuri kembali ke sumbernya.

Keet menempatkan **OWL**, *reasoning*, metode pengembangan ontologi, *top-down development*, *bottom-up development*, dan *text processing* sebagai bagian penting dalam pembelajaran *ontology engineering*. Konsep ini relevan untuk mahasiswa, tetapi dalam buku ini cukup digunakan secara praktis dan ringan. Fokus utamanya bukan membangun ontologi akademik yang rumit, melainkan membuat **peta pengetahuan sederhana** yang membantu corpus RAG menjadi lebih mudah dicari, diuji, dan diaudit.

## 9.13 Menggunakan Ontologi sebagai Metadata Pencarian RAG

Ontologi ringan dapat memperkaya corpus RAG dengan cara sederhana.

Contoh metadata tambahan untuk chunk:

```
{
 "chunk_id": "DOC001-0001",
 "section": "Pengisian KRS",
 "entities": ["Mahasiswa", "KRS"],
 "relations": ["Mahasiswa mengisi KRS"],
 "ontology_terms": ["Aktor", "Dokumen Akademik"]
}
```

Manfaat:

- membantu filtering,
- membantu evaluasi jawaban,
- membantu membuat pertanyaan uji,
- membantu menemukan sinonim,
- membantu menjelaskan relasi antar topik.

Contoh penggunaan:

Kebutuhan RAG	Bantuan Ontologi
Mencari aturan KRS	Temukan relasi terkait KRS
Mengecek jawaban LLM	Bandingkan dengan relasi verified
Membuat pertanyaan uji	Gunakan relasi sebagai basis pertanyaan
Mengelompokkan dokumen	Gunakan class dan concept
Audit corpus	Gunakan evidence dan source

**Ontologi ringan tidak menggantikan vector search. Ia memperkaya konteks.**

## 9.14 Validasi Ontologi Ringan

Sebelum masuk Bab 10, validasi ontologi.

Checklist:

- Apakah setiap relasi berasal dari **relations.csv**?
- Apakah setiap relasi punya evidence?
- Apakah nama subject dan object konsisten?
- Apakah URI tidak mengandung karakter kacau?
- Apakah class terlalu banyak atau terlalu sedikit?
- Apakah relasi tidak terbalik?
- Apakah file **.ttl** bisa dibaca ulang oleh rdflib?
- Apakah diagram Mermaid sesuai dengan relasi CSV?

Contoh relasi terbalik:

```
KRS → menyetujui → Dosen Wali
```

Seharusnya:

```
Dosen Wali → menyetujui → KRS
```

**Kesalahan relasi lebih berbahaya daripada relasi yang belum lengkap.**

## 9.15 Praktik Validasi RDF dengan Python

Buat script:

```
scripts/validate_ontology.py
```

Isi:

```
from pathlib import Path
from rdflib import Graph

TTL_FILE = Path("data/ontology/corpus_ontology.ttl")

def main():
 if not TTL_FILE.exists():
 raise FileNotFoundError(f"File tidak ditemukan: {TTL_FILE}")

 graph = Graph()
 graph.parse(TTL_FILE, format="turtle")

 print(f"File Turtle valid: {TTL_FILE}")
 print(f"Jumlah triple: {len(graph)}")

 if len(graph) == 0:
 print("PERINGATAN: graph kosong")
 else:
 print("Ontology ringan siap dipakai untuk Bab 10")

if __name__ == "__main__":
 main()
```

Jalankan:

```
python scripts/validate_ontology.py
```

Output:

```
File Turtle valid: data/ontology/corpus_ontology.ttl
Jumlah triple: 42
```

Ontology ringan siap dipakai untuk Bab 10

## 9.16 SPARQL Sekilas

Bab ini tidak perlu membahas SPARQL terlalu dalam. Namun mahasiswa perlu tahu bahwa RDF graph dapat di-query.

Contoh query sederhana:

```
SELECT ?object
WHERE {
 ex:Mahasiswa ex:mengisi ?object .
}
```

SPARQL adalah bahasa query untuk RDF; W3C mendefinisikan SPARQL 1.1 sebagai bahasa untuk query dan manipulasi konten RDF graph di Web atau RDF store. ([W3C](#))

Untuk buku ini, SPARQL cukup dipakai sebagai gambaran bahwa ontologi ringan dapat ditanya secara terstruktur.

## 9.17 Output Bab 9

Output wajib:

```
data/ontology/corpus_ontology.ttl
data/ontology/corpus_ontology.owl
data/ontology/ontology_diagram.md
```

Output tambahan:

```
data/ontology/ontology_validation.txt
```

Contoh struktur folder:

```
data/ontology/
├── corpus_ontology.ttl
├── corpus_ontology.owl
├── ontology_diagram.md
└── ontology_validation.txt
```

## 9.18 Troubleshooting Ringkas

Masalah	Penyebab Umum	Solusi
File TTL gagal dibaca	Format Turtle rusak	Gunakan <code>rdflib.serialize()</code>
URI aneh	Subject/object mengandung karakter khusus	Perbaiki <code>slugify()</code>
Relasi terbalik	Pola ekstraksi salah	Cek <code>relations.csv</code>
Banyak class tidak jelas	Mapping class terlalu otomatis	Batasi class utama
Evidence hilang	Tidak dimasukkan dari CSV	Tambahkan <code>hasEvidence</code>
Diagram kosong	<code>relations.csv</code> kosong	Kembali ke Bab 8
Ontologi terlalu rumit	Terlalu banyak properti	Fokus pada relasi utama
Protégé sulit membaca file	Format output kurang cocok	Buka <code>.owl</code> atau <code>.ttl</code>

## 9.19 Praktik Utama Bab 9

Mahasiswa harus:

1. Membaca `data/knowledge/relations.csv`.
2. Mengubah relasi menjadi RDF triple.
3. Menambahkan class sederhana.
4. Menambahkan individual.
5. Menambahkan object property.
6. Menyimpan evidence dan source.
7. Membuat file Turtle.
8. Membuat file OWL/RDF XML.
9. Membuat diagram Mermaid.
10. Memvalidasi file Turtle.

Output akhir:

```
data/ontology/corpus_ontology.ttl
data/ontology/corpus_ontology.owl
data/ontology/ontology_diagram.md
```

## Ringkasan Bab 9

Bab ini mengubah relasi hasil ekstraksi menjadi **ontologi ringan**. Mahasiswa belajar membedakan *taxonomy*, *knowledge graph*, dan ontologi; memahami *class*, *instance*, *property*, dan *relation*; lalu memakai `rdflib` untuk membuat RDF/Turtle dan OWL. Ontologi ringan ini tidak dimaksudkan untuk menggantikan RAG, tetapi untuk memperkaya metadata, memperjelas relasi, dan memudahkan evaluasi jawaban.

### Kalimat kunci Bab 9:

**Ontologi ringan membuat corpus tidak hanya bisa dicari, tetapi juga bisa dipahami, diuji, dan dipertanggungjawabkan.**

# BAB 10 — Studi Kasus End-to-End: PDF ke Corpus Siap Upload RAG

**Proyek RAG tidak selesai saat PDF berhasil diupload; proyek selesai saat jawaban LLM dapat diuji, dilacak, dan diperbaiki dari corpus sumbernya.**

## 10.1 Proyek Akhir Mini

Bab 10 adalah bab integrasi. Semua pekerjaan dari Bab 1 sampai Bab 9 digabung menjadi satu proyek utuh:

```
PDF
→ audit kualitas
→ OCR bila perlu
→ ekstraksi Markdown
→ cleaning
→ metadata
→ chunking
→ file upload RAG
→ ekstraksi knowledge
→ ontologi ringan
→ upload ke Knowledge Base
→ evaluasi jawaban
→ revisi corpus
```

Arah buku menegaskan bahwa studi kasus akhir memakai **PDF Panduan Akademik / SOP Kampus** karena mudah dipahami mahasiswa, tidak terlalu sensitif, dan memiliki aktor, proses, dokumen, syarat, jadwal, serta relasi. Output akhirnya berupa folder `final_project/` yang berisi PDF sumber, laporan audit, clean corpus, chunk, entitas, relasi, ontologi, file upload RAG, pertanyaan uji, dan laporan evaluasi.

## 10.2 Peta Halaman Bab 10

Bagian	Fokus	Estimasi
10.1–10.3	Tujuan studi kasus dan desain proyek	1 hlm
10.4–10.7	Eksekusi pipeline PDF ke corpus	3 hlm
10.8–10.10	Knowledge, ontologi ringan, paket RAG	2 hlm
10.11–10.14	Upload, uji 10 pertanyaan, evaluasi	3 hlm
10.15–10.17	Revisi corpus, output akhir, ringkasan	1 hlm
<b>Total</b>		<b>10 hlm</b>

## 10.3 Case Practical: Panduan Akademik / SOP Kampus

Gunakan dokumen yang sederhana.

Contoh dokumen:

- **Panduan Akademik Mahasiswa**
- **SOP Laboratorium**
- **Brosur Layanan Kampus**
- **Aturan Organisasi Mahasiswa**

Dokumen yang cocok untuk mahasiswa:

- 5–30 halaman,
- bahasa Indonesia,
- tidak berisi data pribadi,
- punya heading,
- punya aktor,
- punya proses,
- punya aturan,
- punya relasi.

Contoh relasi yang diharapkan:

Mahasiswa → mengisi → KRS  
Dosen Wali → menyetujui → KRS  
KRS → berisi → Mata Kuliah  
Mata Kuliah → memiliki → SKS

**Jangan mulai dari dokumen ratusan halaman.** Proyek akhir harus selesai, diuji, dan bisa dijelaskan.

## 10.4 Struktur Folder Akhir

Buat folder:

```
final_project/
├── source.pdf
├── pdf_quality_report.csv
├── clean_corpus.md
├── corpus_index.csv
├── chunks.jsonl
├── entities.csv
├── relations.csv
├── corpus_ontology.ttl
├── rag_upload/
│ ├── corpus_for_rag.md
│ ├── corpus_all.jsonl
│ └── metadata.csv
├── test_questions.csv
└── evaluation_report.md
```

Folder ini adalah **bukti kerja akhir**. Semua file penting harus ada di satu tempat.

## 10.5 Alur Praktik End-to-End

### Langkah 1 — Ambil PDF Sumber

Simpan PDF sebagai:

```
final_project/source.pdf
```

Catat sumber dokumen dalam `metadata.csv`.

### Langkah 2 — Audit Kualitas PDF

Gunakan script Bab 3 untuk membuat:

```
final_project/pdf_quality_report.csv
```

Cek:

- apakah PDF *searchable*,
- apakah perlu OCR,
- berapa halaman,
- apakah teks cukup panjang,
- apakah ada masalah scan.

### Langkah 3 — OCR Jika PDF Scan

Jika `needs_ocr = True`, jalankan OCRmyPDF.

OCRmyPDF relevan karena dokumentasi resminya menyebut tool ini membuat PDF scan menjadi PDF yang memiliki lapisan teks sehingga dapat dicari. ([Open WebUI](#))

### Langkah 4 — Ekstrak ke Markdown

Gunakan Docling atau PyMuPDF sesuai hasil audit.

Docling cocok untuk workflow RAG karena mendukung parsing PDF dan ekspor ke Markdown/JSON, sebagaimana ditekankan dalam arah buku sebelumnya.

## Langkah 5 — Bersihkan Teks dan Tambahkan Metadata

Hasil akhir:

```
final_project/clean_corpus.md
final_project/corpus_index.csv
```

Metadata minimal:

```
document_id: "DOC001"
title: "Panduan Akademik Mahasiswa"
source_file: "source.pdf"
language: "id"
document_type: "panduan"
status: "clean"
```

## Langkah 6 — Buat Chunk

Hasil akhir:

```
final_project/chunks.jsonl
```

Format chunk:

```
{"chunk_id": "DOC001-0001", "document_id": "DOC001", "title": "Panduan Akademik Mahasiswa", "section": "KRS", "page_start": 5, "page_end": 6, "text": "Mahasiswa wajib mengisi KRS..."}
```

Dokumen outline menekankan bahwa struktur chunk perlu menyimpan `chunk_id`, `document_id`, `title`, `section`, `page_start`, `page_end`, dan `text` agar konteks dan jejak sumber tetap terjaga.

## 10.6 Membuat File Upload RAG

Buat folder:

```
final_project/rag_upload/
```

Isi:

```
corpus_for_rag.md
corpus_all.jsonl
metadata.csv
```

Fungsi:

File	Fungsi
<code>corpus_for_rag.md</code>	File Markdown bersih untuk upload ke Knowledge Base
<code>corpus_all.jsonl</code>	Chunk terstruktur untuk pipeline teknis
<code>metadata.csv</code>	Metadata audit corpus

Open WebUI mendukung pembuatan *Knowledge Base* melalui Workspace > Knowledge, upload file atau dokumen, lalu knowledge dapat dilampirkan ke model atau direferensikan dalam chat dengan #. ([Open WebUI](#)) Open WebUI juga menjelaskan bahwa fitur *Knowledge & RAG* memungkinkan AI mengambil informasi dari dokumen melalui *vector search* atau *full-content injection*. ([Open WebUI](#))

## 10.7 Ekstrak Entitas dan Relasi

Dari `corpus_all.jsonl`, buat:

```
final_project/entities.csv
final_project/relations.csv
```

Contoh `entities.csv`:

```
entity,entity_type,source,chunk_id,section
Mahasiswa,DOMAIN_TERM,DOC001,DOC001-0001,Pengisian KRS
KRS,DOMAIN_TERM,DOC001,DOC001-0001,Pengisian KRS
Dosen Wali,DOMAIN_TERM,DOC001,DOC001-0002,Persetujuan KRS
```

Contoh `relations.csv`:

```
subject,relation,object,evidence,source,chunk_id,status
Mahasiswa,mengisi,KRS,"Mahasiswa wajib mengisi
KRS...",DOC001,DOC001-0001,verified
Dosen Wali,menyetujui,KRS,"KRS disetujui oleh Dosen
Wali.",DOC001,DOC001-0002,verified
KRS,berisi,Mata Kuliah,"KRS berisi daftar Mata
Kuliah.",DOC001,DOC001-0003,verified
```

**Aturan keras:** relasi tanpa *evidence* tidak boleh masuk final.

Oltramari dkk. menekankan pentingnya *traceability* melalui metadata agar informasi dapat ditelusuri kembali ke lokasi sumber aslinya; prinsip ini sangat relevan untuk corpus RAG yang harus bisa diaudit.

## 10.8 Buat Ontologi Ringan

Dari `relations.csv`, buat:

```
final_project/corpus_ontology.ttl
```

Contoh Turtle:

```
@prefix ex: <http://example.org/corpus#> .

ex:Mahasiswa ex:mengisi ex:KRS .
ex:DosenWali ex:menyetujui ex:KRS .
ex:KRS ex:berisi ex:MataKuliah .
```

Ontologi ringan ini bukan pengganti RAG. Fungsinya:

- memberi peta relasi,
- membantu evaluasi jawaban,
- membantu membuat pertanyaan uji,
- membantu validasi struktur pengetahuan.

Keet menjelaskan bahwa lapisan pengetahuan dapat membuat data lebih bermakna dan memungkinkan query yang lebih cerdas dibanding hanya membaca data mentah; prinsip ini cocok dipakai secara ringan untuk corpus RAG mahasiswa.

## 10.9 Upload ke Open WebUI Knowledge Base

Alur umum:

1. Buka Open WebUI.
2. Masuk ke **Workspace**.
3. Pilih **Knowledge**.
4. Buat *Knowledge Base* baru.
5. Beri nama, misalnya:

`Corpus Panduan Akademik`

6. Upload:

`final_project/rag_upload/corpus_for_rag.md`

7. Setelah proses selesai, gunakan knowledge dengan # di chat.

Dokumentasi Open WebUI menyebut file yang diupload ke *Knowledge Base* akan diindeks; *Knowledge Base* juga dapat digunakan dalam mode *Full Context* jika dibutuhkan. ([Open WebUI](#))

**Catatan penting:** setelah upload, jangan langsung percaya jawabannya. Uji dulu.

## 10.10 Membuat 10 Pertanyaan Uji

Buat file:

```
final_project/test_questions.csv
```

Contoh:

```
question_id,question,target_answer,expected_source
Q001,Siapa yang menyetujui KRS mahasiswa?,Dosen Wali,DOC001
Q002,Apa isi KRS?,Daftar mata kuliah,DOC001
Q003,Kapan mahasiswa mengisi KRS?,Sesuai jadwal
akademik,DOC001
Q004,Dokumen apa yang menjadi bukti pengambilan mata
kuliah?,KRS,DOC001
Q005,Apa relasi antara mahasiswa KRS dan mata
kuliah?,Mahasiswa mengisi KRS; KRS berisi mata kuliah,DOC001
Q006,Siapa aktor utama dalam proses KRS?,Mahasiswa dan Dosen
Wali,DOC001
Q007,Apa syarat umum pengisian KRS?,Status aktif dan
mengikuti jadwal akademik,DOC001
Q008,Apa yang terjadi jika KRS belum disetujui?,Perlu
persetujuan Dosen Wali,DOC001
Q009,Apa hubungan KRS dengan SKS?,Mata kuliah dalam KRS
memiliki SKS,DOC001
Q010,Bagian dokumen mana yang menjelaskan pengisian
KRS?,Section Pengisian KRS,DOC001
```

Jenis pertanyaan:

- pertanyaan faktual,
- pertanyaan relasi,
- pertanyaan sumber,
- pertanyaan ringkasan pendek,
- pertanyaan lintas konsep.

## 10.11 Prompt Uji di Open WebUI

Gunakan prompt yang memaksa jawaban berbasis corpus.

Gunakan hanya informasi dari Knowledge Base yang saya pilih.

Jawab pertanyaan berikut:

[masukkan pertanyaan]

Aturan:

1. Jangan menambah informasi dari luar corpus.
2. Jika jawaban tidak ditemukan, tulis: "Tidak ditemukan dalam corpus."
3. Sebutkan bagian atau istilah sumber jika tersedia.
4. Jawab singkat dan jelas.

**Tujuan uji bukan membuat jawaban panjang. Tujuan uji adalah memeriksa apakah retrieval tepat.**

## 10.12 Mencatat Hasil Evaluasi

Buat:

```
final_project/evaluation_report.md
```

Format evaluasi:

```
Evaluation Report

Ringkasan
Jumlah pertanyaan: 10
Benar: 7
Salah: 2
Tidak ditemukan: 1

Detail Evaluasi

| ID | Pertanyaan | Target | Jawaban LLM | Status | Catatan |
|---|---|---|---|---|---|
| Q001 | Siapa yang menyetujui KRS? | Dosen Wali | Dosen Wali benar | OK |
| Q002 | Apa isi KRS? | Daftar mata kuliah | Mata kuliah benar | Cukup |
| Q003 | Kapan mahasiswa mengisi KRS? | Sesuai jadwal akademik | Tidak ditemukan | tidak ditemukan | Chunk jadwal kurang jelas |
```

Status:

Status	Arti
benar	Jawaban sesuai corpus
salah	Jawaban bertentangan atau tidak sesuai
tidak ditemukan	Sistem tidak menemukan jawaban
sebagian	Jawaban sebagian benar tetapi kurang lengkap

## 10.13 Script Membuat Template Evaluasi

Buat file:

```
scripts/create_evaluation_template.py
```

Isi:

```
from pathlib import Path
import pandas as pd

QUESTIONS_FILE = Path("final_project/test_questions.csv")
OUTPUT_FILE = Path("final_project/evaluation_template.csv")

def main():
 if not QUESTIONS_FILE.exists():
 raise FileNotFoundError(f"File tidak ditemukan: {QUESTIONS_FILE}")

 df = pd.read_csv(QUESTIONS_FILE)

 df["llm_answer"] = ""
 df["status"] = ""
 df["retrieved_source"] = ""
 df["notes"] = ""

 df.to_csv(OUTPUT_FILE, index=False, encoding="utf-8")

 print(f"Template evaluasi dibuat: {OUTPUT_FILE}")

if __name__ == "__main__":
 main()
```

Jalankan:

```
python scripts/create_evaluation_template.py
```

Output:

```
final_project/evaluation_template.csv
```

## 10.14 Script Mengecek Kelengkapan Final Project

Buat file:

```
scripts/check_final_project.py
```

Isi:

```
from pathlib import Path

REQUIRED_FILES = [
 "final_project/source.pdf",
 "final_project/pdf_quality_report.csv",
 "final_project/clean_corpus.md",
 "final_project/corpus_index.csv",
 "final_project/chunks.jsonl",
 "final_project/entities.csv",
 "final_project/relations.csv",
 "final_project/corpus_ontology.ttl",
 "final_project/rag_upload/corpus_for_rag.md",
 "final_project/rag_upload/corpus_all.jsonl",
 "final_project/rag_upload/metadata.csv",
 "final_project/test_questions.csv",
 "final_project/evaluation_report.md",
]

def main():
 missing = []

 for file_name in REQUIRED_FILES:
 file_path = Path(file_name)

 if file_path.exists():
 print(f"OK: {file_name}")
 else:
 print(f"MISSING: {file_name}")
 missing.append(file_name)

 print("\nRingkasan:")
 print(f"Total file wajib: {len(REQUIRED_FILES)}")
 print(f"File hilang: {len(missing)}")

 if missing:
 print("\nPerbaiki file yang hilang sebelum proyek dinilai.")
 else:
 print("\nFinal project lengkap dan siap dikumpulkan.")

if __name__ == "__main__":
 main()
```

Jalankan:

```
python scripts/check_final_project.py
```

## 10.15 Cara Merevisi Corpus Berdasarkan Hasil Uji

Jika jawaban salah, jangan langsung mengganti model. Periksa corpus.

### Masalah 1 — Jawaban Tidak Ditemukan

Kemungkinan sebab:

- chunk terlalu kecil,
- heading tidak jelas,
- istilah tidak konsisten,
- pertanyaan memakai sinonim yang tidak ada di corpus.

Solusi:

- perbaiki heading,
- tambah sinonim di metadata,
- gabungkan chunk yang terlalu kecil,
- tambahkan section yang lebih jelas.

### Masalah 2 — Jawaban Salah

Kemungkinan sebab:

- chunk salah terambil,
- teks sumber ambigu,
- relasi salah,
- metadata lemah.

Solusi:

- cek `chunks.jsonl`,
- cek `relations.csv`,
- perbaiki evidence,
- revisi clean corpus.

## Masalah 3 — Jawaban Terlalu Panjang

Kemungkinan sebab:

- chunk terlalu besar,
- corpus berisi noise,
- prompt kurang ketat.

Solusi:

- kurangi ukuran chunk,
- bersihkan noise,
- perjelas prompt uji.

**Prinsip tajam:** jika RAG salah, audit dulu corpus, bukan langsung menyalahkan LLM.

## 10.16 Kriteria Keberhasilan Proyek

Proyek dianggap berhasil jika:

- **PDF sudah diaudit,**
- **OCR dilakukan jika perlu,**
- **clean corpus dapat dibaca manusia,**
- **chunk punya metadata lengkap,**
- **file upload RAG tersedia,**
- **entitas dan relasi punya evidence,**
- **ontologi ringan berhasil dibuat,**
- **Knowledge Base dapat menjawab pertanyaan uji,**
- **hasil evaluasi dicatat,**
- **revisi corpus dilakukan berdasarkan bukti.**

Target minimal:

```
10 pertanyaan uji
≥ 7 jawaban benar
0 relasi tanpa evidence
0 chunk tanpa document_id
0 file wajib hilang
```

## 10.17 Output Akhir Proyek

Output akhir wajib:

```
final_project/
├── source.pdf
├── pdf_quality_report.csv
├── clean_corpus.md
├── corpus_index.csv
├── chunks.jsonl
├── entities.csv
├── relations.csv
├── corpus_ontology.ttl
├── rag_upload/
│ ├── corpus_for_rag.md
│ ├── corpus_all.jsonl
│ └── metadata.csv
├── test_questions.csv
└── evaluation_report.md
```

Output tambahan yang disarankan:

```
final_project/
├── ontology_diagram.md
├── evidence_table.csv
├── evaluation_template.csv
└── revision_notes.md
```

## 10.18 Template `revision_notes.md`

```
Revision Notes

Masalah 1
Pertanyaan: Kapan mahasiswa mengisi KRS?
Status awal: Tidak ditemukan

Analisis
Chunk tentang jadwal KRS terlalu pendek dan heading kurang
jelas.

Revisi
- Mengubah heading menjadi "Jadwal Pengisian KRS".
- Menggabungkan paragraf jadwal dengan aturan KRS.
- Membuat ulang chunks.jsonl.
- Upload ulang corpus_for_rag.md.

Status Setelah Revisi
Jawaban ditemukan dan sesuai target.
```

## Ringkasan Bab 10

Bab ini menutup buku dengan proyek utuh: PDF Panduan Akademik / SOP Kampus diubah menjadi **corpus siap upload RAG**, diperkaya dengan entitas, relasi, evidence, dan ontologi ringan, lalu diuji di Open WebUI Knowledge Base. Mahasiswa tidak hanya belajar membuat file, tetapi juga belajar mengevaluasi dan memperbaiki corpus berdasarkan hasil tanya-jawab.

### Kalimat kunci Bab 10:

**RAG yang baik bukan yang sekali upload langsung dipercaya, tetapi yang diuji, dilacak, diperbaiki, dan dibangun dari corpus yang benar.**

# Lampiran A — Checklist PDF ke Corpus RAG

**Jangan upload corpus sebelum lolos audit kualitas, metadata, chunk, dan sumber halaman.**

Checklist ini dipakai untuk memastikan PDF sudah layak diubah menjadi **corpus RAG**.

Area Pemeriksaan	Checklist	Status	Catatan
Tipe PDF	PDF dapat dibuka tanpa error	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	PDF <i>searchable</i> atau hasil scan sudah diketahui	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Jumlah halaman sudah dicatat	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	PDF tidak rusak / terenkripsi / kosong	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
Kualitas OCR	PDF scan sudah diproses OCR	<input type="checkbox"/> OK <input type="checkbox"/> Tidak perlu	
	Bahasa OCR sesuai, misalnya Indonesia / Inggris	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Teks hasil OCR bisa dicari	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Tidak banyak karakter rusak akibat OCR	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Nama, angka, istilah teknis dicek manual	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
Kualitas Markdown	Judul dokumen terbaca jelas	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Heading dan subheading rapi	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Paragraf tidak pecah berlebihan	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Tabel penting tidak berubah menjadi teks kacau	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Header, footer, nomor halaman, watermark sudah dibersihkan	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
Metadata	<code>document_id</code> tersedia	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	<code>title</code> tersedia	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	<code>source_file</code> tersedia	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	<code>language</code> tersedia	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	<code>document_type</code> tersedia	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	<code>page_start</code> dan <code>page_end</code> tersedia bila memungkinkan	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	<code>extraction_tool</code> tercatat	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	

	status jelas: <code>raw</code> , <code>clean</code> , <code>chunked</code> , atau <code>ready_for_rag</code>	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
<b>Chunk</b>	Setiap chunk memiliki <code>chunk_id</code>	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Setiap chunk memiliki <code>document_id</code>	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Setiap chunk memiliki <code>section</code>	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Teks chunk tidak terlalu pendek	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Teks chunk tidak terlalu panjang	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Chunk masih bisa menjawab pertanyaan dasar	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Format JSONL valid	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
<b>Duplikasi</b>	Tidak ada dokumen ganda	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Tidak ada chunk identik berulang tanpa alasan	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Header-footer berulang tidak masuk ke banyak chunk	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Overlap tidak terlalu besar	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
<b>Sumber Halaman</b>	Sumber PDF asli masih tersimpan	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Halaman sumber dicatat bila tersedia	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Evidence relasi dapat ditelusuri ke chunk / halaman	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Jawaban RAG dapat dilacak ke dokumen sumber	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
<b>Kesiapan Upload</b>	<code>corpus_for_rag.md</code> tersedia	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	<code>corpus_all.jsonl</code> tersedia	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	<code>metadata.csv</code> tersedia	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	<code>upload_readme.md</code> tersedia	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Paket upload berada di folder <code>rag_upload/</code>	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	
	Corpus siap diuji dengan pertanyaan evaluasi	<input type="checkbox"/> OK <input type="checkbox"/> Perlu cek	

#### Keputusan akhir:

- ☐ READY\_UPLOAD
- ☐ NEEDS\_CLEANING
- ☐ NEEDS\_RECHUNKING
- ☐ NEEDS\_OCR
- ☐ NEEDS\_MANUAL\_REVIEW
- ☐ REJECTED

# Lampiran B — Template Metadata Corpus

## Golden rule:

**Metadata adalah alamat pengetahuan; tanpa metadata, jawaban RAG sulit diaudit.**

Template ini dipakai untuk membuat `metadata.csv` atau `corpus_index.csv`.

## Kolom Wajib

```
document_id,title,source_file,year,language,document_type,page_start,page_end,extraction_tool,status
```

## Contoh Metadata Corpus

```
document_id,title,source_file,year,language,document_type,page_start,page_end,extraction_tool,status
DOC001,Panduan Akademik
Mahasiswa,panduan-akademik.pdf,2026,id,panduan,1,32,docling,clean
DOC002,SOP
Laboratorium,sop-laboratorium.pdf,2026,id,SOP,1,12,ocrmypdf+pymupdf,chunked
DOC003,Brosur Layanan
Kampus,brosur-layanan.pdf,2026,id,brosur,1,4,pymupdf,ready_for_rag
```

## Penjelasan Kolom

Kolom	Penjelasan
<code>document_id</code>	ID unik dokumen, misalnya <code>DOC001</code> .
<code>title</code>	Judul dokumen.
<code>source_file</code>	Nama file PDF sumber.
<code>year</code>	Tahun dokumen.
<code>language</code>	Bahasa dokumen, misalnya <code>id</code> atau <code>en</code> .
<code>document_type</code>	Jenis dokumen: panduan, SOP, brosur, regulasi, laporan.
<code>page_start</code>	Halaman awal yang diproses.
<code>page_end</code>	Halaman akhir yang diproses.
<code>extraction_tool</code>	Tool ekstraksi, misalnya <code>docling</code> , <code>pymupdf</code> , <code>pdfplumber</code> , <code>ocrmypdf+pymupdf</code> .
<code>status</code>	Status dokumen: <code>raw</code> , <code>ocr_done</code> , <code>markdown</code> , <code>clean</code> , <code>chunked</code> , <code>ready_for_rag</code> .

## Status yang Disarankan

Status	Arti
<code>raw</code>	PDF baru dikumpulkan, belum diproses.
<code>ocr_done</code>	PDF scan sudah di-OCR.
<code>markdown</code>	PDF sudah dikonversi ke Markdown.
<code>clean</code>	Teks sudah dibersihkan.
<code>chunked</code>	Corpus sudah dipotong menjadi chunk.
<code>ready_for_rag</code>	File siap upload ke RAG.
<code>needs_review</code>	Perlu pemeriksaan manual.
<code>rejected</code>	Tidak layak dipakai.

# Lampiran C — Template Chunk RAG

## Golden rule:

**RAG tidak mencari dokumen; RAG mencari chunk. Maka setiap chunk harus jelas sumber, konteks, dan isi teksnya.**

Template ini dipakai untuk membuat `chunks.jsonl` atau `corpus_all.jsonl`.

## Kolom Wajib

```
chunk_id,document_id,title,section,page_start,page_end,text
```

## Contoh CSV Chunk

```
chunk_id,document_id,title,section,page_start,page_end,text
DOC001-0001,DOC001,Panduan Akademik Mahasiswa,Pengisian
KRS,5,6,"Mahasiswa wajib mengisi KRS sebelum perkuliahan
dimulai."
DOC001-0002,DOC001,Panduan Akademik Mahasiswa,Persetujuan
Dosen Wali,6,6,"KRS harus disetujui oleh Dosen Wali."
DOC001-0003,DOC001,Panduan Akademik Mahasiswa,Isi
KRS,7,7,"KRS berisi daftar mata kuliah yang diambil
mahasiswa."
```

## Contoh JSONL Chunk

```
{"chunk_id":"DOC001-0001","document_id":"DOC001","title":"Pan
duan Akademik Mahasiswa","section":"Pengisian
KRS","page_start":5,"page_end":6,"text":"Mahasiswa wajib
mengisi KRS sebelum perkuliahan dimulai."}
{"chunk_id":"DOC001-0002","document_id":"DOC001","title":"Pan
duan Akademik Mahasiswa","section":"Persetujuan Dosen
Wali","page_start":6,"page_end":6,"text":"KRS harus disetujui
oleh Dosen Wali."}
{"chunk_id":"DOC001-0003","document_id":"DOC001","title":"Pan
duan Akademik Mahasiswa","section":"Isi
KRS","page_start":7,"page_end":7,"text":"KRS berisi daftar
mata kuliah yang diambil mahasiswa."}
```

## Penjelasan Kolom

Kolom	Penjelasan
<code>chunk_id</code>	ID unik chunk, misalnya <code>DOC001-0001</code> .
<code>document_id</code>	ID dokumen asal.
<code>title</code>	Judul dokumen.
<code>section</code>	Bagian / heading asal chunk.
<code>page_start</code>	Halaman awal sumber chunk.
<code>page_end</code>	Halaman akhir sumber chunk.
<code>text</code>	Isi chunk yang akan dicari oleh RAG.

## Checklist Chunk

- Tidak terlalu pendek.
- Tidak terlalu panjang.
- Tidak memotong kalimat penting.
- Memiliki section yang jelas.
- Memiliki sumber halaman bila tersedia.
- Teks chunk dapat menjawab pertanyaan dasar.
- Tidak berisi header-footer berulang.

## Lampiran D — Template Entity dan Relation Table

### Golden rule:

**Relasi tanpa evidence bukan knowledge; itu hanya dugaan.**

Template ini dipakai untuk membuat `relations.csv` dan `evidence_table.csv`.

Struktur ini mengikuti arah buku bahwa corpus RAG dapat diperkaya dengan **entitas, konsep, relasi, dan evidence**, sehingga lebih mudah diuji dan dikembangkan.

### Kolom Wajib

```
subject,relation,object,evidence,source_file,page,confidence
```

### Contoh Entity dan Relation Table

```
subject,relation,object,evidence,source_file,page,confidence
Mahasiswa,mengisi,KRS,"Mahasiswa wajib mengisi KRS sebelum
perkuliahan dimulai.",panduan-akademik.pdf,5,0.95
Dosen Wali,menyetujui,KRS,"KRS harus disetujui oleh Dosen
Wali.",panduan-akademik.pdf,6,0.95
KRS,berisi,Mata Kuliah,"KRS berisi daftar mata kuliah yang
diambil mahasiswa.",panduan-akademik.pdf,7,0.90
Mata Kuliah,memiliki,SKS,"Setiap mata kuliah memiliki jumlah
SKS tertentu.",panduan-akademik.pdf,8,0.85
```

### Penjelasan Kolom

Kolom	Penjelasan
<code>subject</code>	Entitas / konsep pertama.
<code>relation</code>	Hubungan antara subject dan object.
<code>object</code>	Entitas / konsep kedua.
<code>evidence</code>	Kalimat sumber yang mendukung relasi.
<code>source_file</code>	PDF asal.
<code>page</code>	Halaman sumber evidence.
<code>confidence</code>	Tingkat keyakinan awal, misalnya <code>0.70–1.00</code> .

## Skala Confidence

Nilai	Arti
0.95–1.00	Relasi eksplisit, evidence jelas.
0.80–0.94	Relasi kuat, tetapi perlu cek ringan.
0.60–0.79	Relasi kandidat, perlu validasi manual.
<0.60	Jangan dipakai sebelum diperiksa.

## Aturan Validasi

- Evidence wajib ada.
- Subject dan object tidak boleh tertukar.
- Relasi harus eksplisit dalam teks.
- Confidence rendah harus dicek manual.
- Relasi yang tidak ditemukan dalam corpus harus dibuang.

# Lampiran E — 30 Prompt AI untuk Membuat Corpus RAG

## Golden rule:

**Gunakan AI untuk membantu ekstraksi dan evaluasi, bukan untuk mengarang pengetahuan baru.**

Prompt berikut dipakai untuk membantu proses PDF menjadi corpus RAG. Semua prompt harus dijalankan dengan prinsip: **jawaban harus berbasis teks sumber, tidak boleh menambah informasi dari luar corpus.**

## A. Prompt Cek Kualitas Teks

### 1. Cek noise OCR

Periksa teks berikut dan identifikasi kemungkinan noise OCR.

Aturan:

1. Jangan mengubah makna.
2. Tandai kata yang terlihat rusak.
3. Berikan usulan koreksi hanya jika sangat jelas.
4. Jangan menambah informasi baru.

Teks:

[masukkan teks]

### 2. Cek header-footer berulang

Cari pola header, footer, nomor halaman, atau watermark yang berulang dalam teks berikut.

Output:

- Pola berulang
- Dugaan jenis noise
- Rekomendasi penghapusan

Teks:

[masukkan teks]

### 3. Cek struktur heading

Periksa apakah teks berikut memiliki struktur heading yang jelas.

Output:

1. Heading yang terdeteksi.
2. Heading yang tampak rusak.
3. Usulan struktur heading Markdown.
4. Jangan mengubah isi utama.

Teks:

[masukkan teks]

### 4. Cek kelayakan corpus

Nilai apakah teks berikut layak menjadi clean corpus untuk RAG.

Gunakan kriteria:

- kebersihan teks,
- struktur,
- paragraf,
- noise,
- metadata,
- kesiapan chunking.

Output dalam tabel:

aspek, status, catatan, rekomendasi.

Teks:

[masukkan teks]

## B. Prompt Ekstraksi Konsep

### 5. Ekstraksi konsep utama

Ekstrak konsep utama dari teks berikut.

Aturan:

1. Ambil hanya konsep yang muncul eksplisit.
2. Jangan menambah konsep dari luar.
3. Gabungkan istilah yang sama.
4. Beri alasan singkat.

Format:

concept, evidence, reason

Teks:

[masukkan teks]

### 6. Ekstraksi istilah domain

Ambil istilah domain penting dari teks berikut.

Kategori:

- aktor,
- dokumen,
- proses,
- syarat,
- jadwal,
- unit kerja,
- istilah teknis.

Output:

term, category, evidence

Teks:

[masukkan teks]

### 7. Normalisasi konsep

Normalisasi daftar konsep berikut.

Tugas:

1. Gabungkan istilah yang sama.
2. Rapiakan kapitalisasi.
3. Tandai sinonim.
4. Jangan menghapus konsep penting.

Daftar konsep:

[masukkan daftar]

## 8. Buat daftar konsep prioritas

Dari konsep berikut, pilih konsep yang paling penting untuk corpus RAG.

Kriteria:

- sering muncul,
- penting untuk menjawab pertanyaan,
- memiliki relasi dengan konsep lain,
- relevan dengan domain dokumen.

Output:

concept, priority, reason

Daftar konsep:

[masukkan daftar]

## C. Prompt Ekstraksi Entitas

### 9. Ekstraksi entitas dari chunk

Ekstrak entitas dari chunk berikut.

Kategori entitas:

- orang/peran,
- dokumen,
- unit organisasi,
- proses,
- istilah akademik,
- waktu/jadwal.

Aturan:

1. Hanya ambil entitas yang muncul dalam teks.
2. Sertakan evidence.
3. Jangan menebak.

Format:

entity, entity\_type, evidence

Chunk:

[masukkan chunk]

## 10. Ekstraksi aktor

Identifikasi aktor yang disebut dalam teks berikut.

Output:

aktor, peran, evidence

Aturan:

- Jangan menambah aktor dari luar teks.
- Jika peran tidak jelas, tulis "tidak jelas".

Teks:

[masukkan teks]

## 11. Ekstraksi dokumen dan artefak

Identifikasi dokumen, formulir, atau artefak yang disebut dalam teks.

Output:

nama\_dokumen, fungsi, evidence

Teks:

[masukkan teks]

## 12. Validasi entitas

Validasi daftar entitas berikut berdasarkan teks sumber.

Tugas:

1. Tandai valid / tidak valid.
2. Berikan evidence untuk entitas valid.
3. Berikan alasan untuk entitas tidak valid.

Teks sumber:

[masukkan teks]

Daftar entitas:

[masukkan daftar entitas]

## D. Prompt Ekstraksi Relasi

### 13. Ekstraksi relasi eksplisit

Ekstrak relasi eksplisit dari teks berikut.

Aturan:

1. Hanya ambil relasi yang tertulis jelas.
2. Setiap relasi wajib punya evidence.
3. Jangan menyimpulkan terlalu jauh.
4. Jangan memakai pengetahuan luar.

Format:

subject, relation, object, evidence

Teks:

[masukkan teks]

### 14. Ekstraksi relasi aktor-proses

Cari relasi antara aktor dan proses dalam teks berikut.

Format:

actor, action, process\_or\_object, evidence

Teks:

[masukkan teks]

### 15. Ekstraksi relasi dokumen-proses

Cari relasi antara dokumen dan proses dalam teks berikut.

Contoh relasi:

- digunakan untuk
- berisi
- menjadi syarat
- disetujui oleh
- diajukan oleh

Format:

subject, relation, object, evidence

Teks:

[masukkan teks]

## 16. Ekstraksi relasi syarat

Ekstrak relasi syarat dari teks berikut.

Format:

process, requires, requirement, evidence

Aturan:

- Ambil hanya syarat yang eksplisit.
- Jangan menambah syarat dari asumsi.

Teks:

[masukkan teks]

## 17. Validasi relasi

Validasi relasi berikut berdasarkan teks sumber.

Tugas:

1. Tandai valid, lemah, atau tidak valid.
2. Cek apakah evidence benar-benar mendukung relasi.
3. Perbaiki subject/object jika tertukar.
4. Jangan menambah relasi baru kecuali jelas dari teks.

Teks sumber:

[masukkan teks]

Daftar relasi:

[masukkan relasi]

## E. Prompt Pembuatan Metadata

### 18. Buat metadata dokumen

Buat metadata dokumen dari informasi berikut.

Format:

document\_id, title, source\_file, year, language,  
document\_type, page\_start, page\_end, extraction\_tool, status

Informasi:

[masukkan informasi dokumen]

### 19. Buat YAML front matter

Buat YAML front matter untuk corpus Markdown berikut.

Kolom wajib:

document\_id, title, source\_file, year, language,  
document\_type, page\_start, page\_end, extraction\_tool, status

Informasi dokumen:

[masukkan informasi]

### 20. Cek konsistensi metadata

Periksa konsistensi metadata berikut.

Cek:

- document\_id,
- title,
- source\_file,
- year,
- language,
- document\_type,
- page\_start,
- page\_end,
- extraction\_tool,
- status.

Output:

field, status, catatan

Metadata:

[masukkan metadata]

## F. Prompt Validasi Chunk

### 21. Validasi kualitas chunk

Nilai kualitas chunk berikut untuk RAG.

Kriteria:

- konteks cukup,
- tidak terlalu pendek,
- tidak terlalu panjang,
- section jelas,
- bisa menjawab pertanyaan,
- tidak ada noise.

Output:

aspek, status, catatan

Chunk:

[masukkan chunk]

### 22. Cek apakah chunk bisa menjawab pertanyaan

Berdasarkan chunk berikut, buat 3 pertanyaan yang bisa dijawab langsung dari chunk.

Aturan:

1. Pertanyaan harus faktual.
2. Jawaban harus ada dalam chunk.
3. Jangan membuat pertanyaan yang butuh pengetahuan luar.

Chunk:

[masukkan chunk]

### 23. Cek chunk terlalu kecil / besar

Evaluasi apakah chunk berikut terlalu kecil, terlalu besar, atau sudah cukup.

Output:

- status ukuran,
- masalah konteks,
- rekomendasi pemotongan atau penggabungan.

Chunk:

[masukkan chunk]

## 24. Buat ringkasan chunk

Buat ringkasan singkat dari chunk berikut.

Aturan:

1. Maksimal 3 kalimat.
2. Jangan menambah informasi baru.
3. Sertakan istilah utama.

Chunk:

[masukkan chunk]

## G. Prompt Pembuatan Pertanyaan Uji

### 25. Buat pertanyaan uji faktual

Buat 10 pertanyaan uji faktual dari corpus berikut.

Aturan:

1. Jawaban harus tersedia dalam corpus.
2. Sertakan target jawaban.
3. Sertakan section sumber jika tersedia.
4. Jangan membuat pertanyaan opini.

Format:

question\_id, question, target\_answer, source\_section

Corpus:

[masukkan corpus]

### 26. Buat pertanyaan uji relasi

Buat pertanyaan uji berbasis relasi dari daftar relasi berikut.

Format:

question, target\_answer, relation\_source

Daftar relasi:

[masukkan relations.csv]

## 27. Buat pertanyaan uji tidak ditemukan

Buat 5 pertanyaan yang seharusnya dijawab "Tidak ditemukan dalam corpus".

Aturan:

1. Pertanyaan masih relevan dengan topik.
2. Jawaban tidak boleh ada dalam corpus.
3. Tujuannya menguji apakah RAG berani mengatakan tidak tahu.

Corpus:

[masukkan corpus]

## H. Prompt Evaluasi Jawaban RAG

### 28. Evaluasi jawaban RAG

Evaluasi jawaban RAG berikut berdasarkan target jawaban dan evidence.

Output:

- status: benar / sebagian / salah / tidak ditemukan
- alasan
- rekomendasi perbaikan corpus

Pertanyaan:

[masukkan pertanyaan]

Target jawaban:

[masukkan target]

Jawaban RAG:

[masukkan jawaban]

Evidence:

[masukkan evidence]

## 29. Deteksi halusinasi jawaban

Cek apakah jawaban berikut mengandung informasi yang tidak ada dalam corpus.

Aturan:

1. Tandai klaim yang didukung corpus.
2. Tandai klaim yang tidak didukung corpus.
3. Jangan memakai pengetahuan luar.
4. Berikan rekomendasi revisi.

Corpus:

[masukkan corpus]

Jawaban:

[masukkan jawaban]

## 30. Rekomendasi perbaikan corpus

Berdasarkan hasil evaluasi berikut, berikan rekomendasi perbaikan corpus.

Cek:

- cleaning,
- metadata,
- chunking,
- relasi,
- evidence,
- pertanyaan uji.

Hasil evaluasi:

[masukkan evaluation\_report]

Lampiran ini adalah alat kerja praktis. Mahasiswa tidak hanya belajar membuat corpus, tetapi juga belajar **mengaudit, menstrukturkan, menguji, dan memperbaiki corpus**. Prinsip utamanya tetap sama: **RAG yang kuat bukan dimulai dari model yang besar, tetapi dari corpus yang benar, bersih, ber-metadata, dan dapat ditelusuri ke sumber.**

# Lampiran F — Script Python Ringkas

## Golden rule:

**Script yang baik bukan sekadar berjalan, tetapi menghasilkan corpus yang rapi, ber-metadata, dapat diaudit, dan siap masuk RAG.**

## F.1 Struktur Folder yang Dipakai

Gunakan struktur berikut agar semua script konsisten.

```
pdf-to-rag-corpus/
├── data/
│ ├── pdf/
│ ├── ocr/
│ ├── raw_text/
│ ├── markdown/
│ ├── clean/
│ ├── chunks/
│ ├── metadata/
│ ├── knowledge/
│ ├── ontology/
│ └── rag_upload/
├── scripts/
└── output/
```

Struktur ini mengikuti arah workspace buku: PDF mentah dipisahkan dari OCR, teks mentah, Markdown, clean corpus, chunk, metadata, knowledge, ontology, dan file final **rag\_upload/**.

## F.2 Instalasi Library Python

Jalankan di dalam *virtual environment*.

```
pip install pymupdf pdfplumber pandas spacy rdflib docling
```

Fungsi singkat:

Library	Fungsi
pymupdf	Audit PDF dan ekstraksi teks
pdfplumber	Ekstraksi teks dan tabel sederhana
pandas	Membuat CSV metadata
spacy	NLP dasar / entitas
rdflib	Membuat RDF/Turtle
docling	Konversi dokumen ke Markdown / JSON

PyMuPDF mendukung ekstraksi teks melalui `Page.get_text()`, OCRmyPDF menambahkan lapisan teks OCR ke PDF scan agar dapat dicari, Docling mendukung parsing PDF dan ekspor ke Markdown/JSON untuk workflow AI/RAG, dan RDFLib dapat menyimpan graph RDF ke Turtle melalui `Graph.serialize()`. ([pymupdf.readthedocs.io](https://pymupdf.readthedocs.io))

## F.3 Script 1 — Cek PDF

**Tujuan:** mengecek apakah PDF memiliki teks, jumlah halaman, dan apakah perlu OCR.

Buat file:

```
scripts/01_check_pdf.py

from pathlib import Path
import fitz
import pandas as pd

PDF_DIR = Path("data/pdf")
OUTPUT_FILE =
Path("data/metadata/pdf_quality_report.csv")

MIN_AVG_TEXT_LENGTH = 50

def check_pdf(pdf_path):
 try:
 doc = fitz.open(pdf_path)
 page_lengths = []

 for page in doc:
 text = page.get_text("text")
 page_lengths.append(len(text.strip()))

 doc.close()

 pages = len(page_lengths)
 total_text = sum(page_lengths)
 avg_text_length = total_text / pages if pages
 else 0

 has_text = total_text > 0
 suspected_scan = avg_text_length <
MIN_AVG_TEXT_LENGTH
needs_ocr = suspected_scan

 notes = []
 if not has_text:
 notes.append("Tidak ada teks terdeteksi")
 if suspected_scan:
 notes.append("Kemungkinan PDF scan atau teks
sangat sedikit")
```

```

 return {
 "filename": pdf_path.name,
 "pages": pages,
 "has_text": has_text,
 "avg_text_length": round(avg_text_length,
2),
 "suspected_scan": suspected_scan,
 "needs_ocr": needs_ocr,
 "notes": "; ".join(notes) if notes else
"OK",
 }

 except Exception as error:
 return {
 "filename": pdf_path.name,
 "pages": 0,
 "has_text": False,
 "avg_text_length": 0,
 "suspected_scan": True,
 "needs_ocr": True,
 "notes": f"ERROR: {error}",
 }

def main():
 OUTPUT_FILE.parent.mkdir(parents=True,
exist_ok=True)

 rows = []
 for pdf_file in sorted(PDF_DIR.glob("*.pdf")):
 rows.append(check_pdf(pdf_file))

 df = pd.DataFrame(rows)
 df.to_csv(OUTPUT_FILE, index=False,
encoding="utf-8")

 print(f"Laporan cek PDF dibuat: {OUTPUT_FILE}")

if __name__ == "__main__":
 main()

```

Jalankan:

```
python scripts/01_check_pdf.py
```

Output:

```
data/metadata/pdf_quality_report.csv
```

## F.4 Script 2 — OCR Batch

**Tujuan:** menjalankan OCR untuk semua PDF yang diduga scan.

Buat file:

```
scripts/02_ocr_batch.py

from pathlib import Path
import subprocess
import pandas as pd

PDF_DIR = Path("data/pdf")
OCR_DIR = Path("data/ocr")
REPORT_FILE =
Path("data/metadata/pdf_quality_report.csv")

LANGUAGE = "ind+eng"

def run_ocr(input_pdf, output_pdf):
 command = [
 "ocrmypdf",
 "-l", LANGUAGE,
 "--deskew",
 "--clean",
 str(input_pdf),
 str(output_pdf),
]

 subprocess.run(command, check=True)

def main():
 if not REPORT_FILE.exists():
 raise FileNotFoundError(f"File report tidak
ditemukan: {REPORT_FILE}")

 OCR_DIR.mkdir(parents=True, exist_ok=True)

 df = pd.read_csv(REPORT_FILE)

 for _, row in df.iterrows():
 filename = row["filename"]
 needs_ocr = bool(row["needs_ocr"])
```

```

 if not needs_ocr:
 print(f"Skip OCR: {filename}")
 continue

 input_pdf = PDF_DIR / filename
 output_pdf = OCR_DIR / filename.replace(".pdf",
 "_ocr.pdf")

 print(f"OCR: {input_pdf} -> {output_pdf}")
 run_ocr(input_pdf, output_pdf)

 print("OCR batch selesai.")

if __name__ == "__main__":
 main()

```

Jalankan:

```
python scripts/02_ocr_batch.py
```

Output:

```
data/ocr/*.pdf
```

OCRmyPDF cocok untuk tahap ini karena dokumentasi resminya menyatakan bahwa tool ini menambahkan *OCR text layer* ke PDF scan sehingga dokumen dapat dicari. ([ocrmypdf.readthedocs.io](https://ocrmypdf.readthedocs.io))

## F.5 Script 3 — Ekstraksi Teks

**Tujuan:** mengekstrak teks dari PDF *searchable* atau hasil OCR.

Buat file:

```
scripts/03_extract_text.py

from pathlib import Path
import fitz

PDF_DIR = Path("data/pdf")
OCR_DIR = Path("data/ocr")
RAW_TEXT_DIR = Path("data/raw_text")

def choose_input_pdf(pdf_file):
 ocr_file = OCR_DIR / pdf_file.name.replace(".pdf",
 "_ocr.pdf")

 if ocr_file.exists():
 return ocr_file

 return pdf_file

def extract_text(pdf_path):
 doc = fitz.open(pdf_path)
 parts = []

 for page_number, page in enumerate(doc, start=1):
 text = page.get_text("text", sort=True)
 parts.append(f"\n\n--- PAGE {page_number}
---\n\n{text}")

 doc.close()
 return "\n".join(parts)

def main():
 RAW_TEXT_DIR.mkdir(parents=True, exist_ok=True)

 for pdf_file in sorted(PDF_DIR.glob("*.pdf")):
 input_pdf = choose_input_pdf(pdf_file)
 output_txt = RAW_TEXT_DIR /
pdf_file.name.replace(".pdf", ".txt")

 text = extract_text(input_pdf)
```

```
output_txt.write_text(text, encoding="utf-8")

print(f"Teks dibuat: {output_txt}")

if __name__ == "__main__":
 main()
```

Jalankan:

```
python scripts/03_extract_text.py
```

Output:

```
data/raw_text/*.txt
```

PyMuPDF dapat mengambil teks halaman melalui `Page.get_text()`, termasuk mode yang membantu mengakses teks, blok, dan kata beserta informasi posisi. ([pymupdf.readthedocs.io](https://pymupdf.readthedocs.io))

## F.6 Script 4 — Konversi Markdown dengan Docling

**Tujuan:** mengubah PDF menjadi Markdown, terutama untuk dokumen yang membutuhkan struktur lebih rapi.

Buat file:

```
scripts/04_convert_markdown.py

from pathlib import Path
import subprocess

PDF_DIR = Path("data/pdf")
OCR_DIR = Path("data/ocr")
MARKDOWN_DIR = Path("data/markdown")

def choose_input_pdf(pdf_file):
 ocr_file = OCR_DIR / pdf_file.name.replace(".pdf",
 "_ocr.pdf")

 if ocr_file.exists():
 return ocr_file

 return pdf_file

def convert_with_docling(input_pdf):
 command = [
 "docling",
 str(input_pdf),
 "--to",
 "md",
 "--output",
 str(MARKDOWN_DIR),
]

 subprocess.run(command, check=True)

def main():
 MARKDOWN_DIR.mkdir(parents=True, exist_ok=True)

 for pdf_file in sorted(PDF_DIR.glob("*.pdf")):
 input_pdf = choose_input_pdf(pdf_file)
 print(f"Konversi Markdown: {input_pdf}")
 convert_with_docling(input_pdf)
```

```
print("Konversi Markdown selesai.")

if __name__ == "__main__":
 main()
```

Jalankan:

```
python scripts/04_convert_markdown.py
```

Output:

```
data/markdown/*.md
```

Docling relevan untuk corpus RAG karena mendukung *advanced PDF understanding, reading order*, struktur tabel, formula, dan ekspor ke Markdown serta JSON. ([docling-project.github.io](https://github.com/docling-project/docling))

## F.7 Script 5 — Cleaning

**Tujuan:** membersihkan Markdown mentah menjadi *clean corpus*.

Buat file:

```
scripts/05_cleaning.py

from pathlib import Path
import re
import pandas as pd

MARKDOWN_DIR = Path("data/markdown")
CLEAN_DIR = Path("data/clean")
METADATA_FILE = Path("data/metadata/corpus_index.csv")

def normalize_text(text):
 text = text.replace("\r\n", "\n")
 text = re.sub(r"\n\s*---\s*PAGE\s+\d+\s*---\s*\n",
"\n", text, flags=re.IGNORECASE)
 text = re.sub(r"\n\s*\d+\s*\n", "\n", text)
 text = re.sub(r"[\t]+", " ", text)
 text = re.sub(r"\n{3,}", "\n\n", text)
 return text.strip()

def build_front_matter(document_id, title, source_file,
year, language, document_type, extraction_tool, status):
 return f"""---
document_id: "{document_id}"
title: "{title}"
source_file: "{source_file}"
year: {year}
language: "{language}"
document_type: "{document_type}"
extraction_tool: "{extraction_tool}"
status: "{status}"

"""

def main():
 CLEAN_DIR.mkdir(parents=True, exist_ok=True)
 METADATA_FILE.parent.mkdir(parents=True,
exist_ok=True)
```

```

rows = []

for index, md_file in
enumerate(sorted(MARKDOWN_DIR.glob("*.md")), start=1):
 document_id = f"DOC{index:03d}"
 title = md_file.stem.replace("_", " ").title()
 source_file = md_file.stem + ".pdf"

 raw_text = md_file.read_text(encoding="utf-8",
errors="ignore")
 clean_text = normalize_text(raw_text)

 clean_file = CLEAN_DIR /
f"{document_id}.clean.md"

 front_matter = build_front_matter(
 document_id=document_id,
 title=title,
 source_file=source_file,
 year=2026,
 language="id",
 document_type="panduan",
 extraction_tool="docling",
 status="clean",
)

 clean_file.write_text(front_matter + clean_text,
encoding="utf-8")

 rows.append({
 "document_id": document_id,
 "title": title,
 "source_file": source_file,
 "year": 2026,
 "language": "id",
 "document_type": "panduan",
 "page_start": 1,
 "page_end": "",
 "extraction_tool": "docling",
 "status": "clean",
 "clean_file": str(clean_file),
 })

 print(f"Clean corpus dibuat: {clean_file}")

pd.DataFrame(rows).to_csv(METADATA_FILE,
index=False, encoding="utf-8")
print(f"Corpus index dibuat: {METADATA_FILE}")

```

```
if __name__ == "__main__":
 main()
```

Jalankan:

```
python scripts/05_cleaning.py
```

Output:

```
data/clean/*.clean.md
data/metadata/corpus_index.csv
```

Cleaning dan metadata adalah inti corpus RAG karena dokumen terlampir menekankan bahwa clean corpus harus memiliki struktur, metadata, dan status yang jelas sebelum masuk tahap *chunking*.

## F.8 Script 6 — Chunking

**Tujuan:** memotong clean corpus menjadi chunk JSONL.

Buat file:

```
scripts/06_chunking.py

from pathlib import Path
import re
import json
import pandas as pd

CLEAN_DIR = Path("data/clean")
CHUNK_FILE = Path("data/chunks/corpus_all.jsonl")
REPORT_FILE = Path("data/metadata/chunk_report.csv")

MAX_CHARS = 1200
OVERLAP_CHARS = 150

def remove_front_matter(text):
 return re.sub(r"^---\n.*?\n---\n", "", text,
flags=re.DOTALL).strip()

def split_by_heading(text):
 sections = []
 current_section = "Tanpa Section"
 current_lines = []

 for line in text.splitlines():
 stripped = line.strip()

 if stripped.startswith("#"):
 if current_lines:
 sections.append((current_section,
"\n".join(current_lines).strip()))
 current_lines = []

 current_section =
stripped.lstrip("#").strip()
 else:
 current_lines.append(line)

 if current_lines:
 sections.append((current_section,
"\n".join(current_lines).strip()))
```

```

 return [(section, content) for section, content in
sections if content]

def split_long_text(text):
 chunks = []
 start = 0

 while start < len(text):
 end = start + MAX_CHARS
 chunk = text[start:end].strip()

 if chunk:
 chunks.append(chunk)

 if end >= len(text):
 break

 start = end - OVERLAP_CHARS

 return chunks

def main():
 CHUNK_FILE.parent.mkdir(parents=True, exist_ok=True)
 REPORT_FILE.parent.mkdir(parents=True,
exist_ok=True)

 all_chunks = []
 report_rows = []
 chunk_counter = 1

 for clean_file in
sorted(CLEAN_DIR.glob("*.clean.md")):
 document_id = clean_file.stem.replace(".clean",
"")

 title = clean_file.stem.replace(".clean",
"").replace("_", " ").title()

 text = clean_file.read_text(encoding="utf-8",
errors="ignore")
 text = remove_front_matter(text)

 sections = split_by_heading(text)

 for section, section_text in sections:
 parts = split_long_text(section_text)

```

```

 for part in parts:
 chunk_id =
f"{document_id}-{chunk_counter:04d}"

 chunk = {
 "chunk_id": chunk_id,
 "document_id": document_id,
 "title": title,
 "section": section,
 "page_start": 1,
 "page_end": 1,
 "text": part,
 }

 all_chunks.append(chunk)

 report_rows.append({
 "chunk_id": chunk_id,
 "document_id": document_id,
 "section": section,
 "char_length": len(part),
 "word_count": len(part.split()),
 })

 chunk_counter += 1

 with CHUNK_FILE.open("w", encoding="utf-8") as file:
 for chunk in all_chunks:
 file.write(json.dumps(chunk,
ensure_ascii=False) + "\n")

 pd.DataFrame(report_rows).to_csv(REPORT_FILE,
index=False, encoding="utf-8")

 print(f"Chunk JSONL dibuat: {CHUNK_FILE}")
 print(f"Chunk report dibuat: {REPORT_FILE}")
 print(f"Jumlah chunk: {len(all_chunks)}")

if __name__ == "__main__":
 main()

```

Jalankan:

```
python scripts/06_chunking.py
```

Output:

```
data/chunks/corpus_all.jsonl
data/metadata/chunk_report.csv
```

Chunk yang baik perlu menyimpan **chunk\_id**, **document\_id**, **title**, **section**, **page\_start**, **page\_end**, dan **text**, sebagaimana ditekankan dalam outline buku.

## F.9 Script 7 — Entity Extraction

**Tujuan:** mengekstrak entitas domain dari chunk.

Buat file:

```
scripts/07_entity_extraction.py

from pathlib import Path
import json
import re
import pandas as pd

CHUNK_FILE = Path("data/chunks/corpus_all.jsonl")
OUTPUT_FILE = Path("data/knowledge/entities.csv")

DOMAIN_ENTITIES = [
 "Mahasiswa",
 "Dosen Wali",
 "KRS",
 "KHS",
 "Mata Kuliah",
 "SKS",
 "Semester",
 "Biro Akademik",
 "Cuti Akademik",
]

def load_chunks():
 chunks = []

 with CHUNK_FILE.open("r", encoding="utf-8") as file:
 for line in file:
 line = line.strip()

 if line:
 chunks.append(json.loads(line))

 return chunks

def main():
 OUTPUT_FILE.parent.mkdir(parents=True,
 exist_ok=True)

 rows = []
 chunks = load_chunks()
```

```

for chunk in chunks:
 text = chunk["text"]

 for entity in DOMAIN_ENTITIES:
 pattern = rf"\b{re.escape(entity)}\b"

 if re.search(pattern, text,
flags=re.IGNORECASE):
 rows.append({
 "entity": entity,
 "entity_type": "DOMAIN_TERM",
 "source": chunk["document_id"],
 "chunk_id": chunk["chunk_id"],
 "section": chunk["section"],
 })

 df = pd.DataFrame(rows).drop_duplicates()
 df.to_csv(OUTPUT_FILE, index=False,
encoding="utf-8")

 print(f"Entities dibuat: {OUTPUT_FILE}")
 print(f"Jumlah entitas ditemukan: {len(df)}")

if __name__ == "__main__":
 main()

```

Jalankan:

```
python scripts/07_entity_extraction.py
```

Output:

```
data/knowledge/entities.csv
```

## F.10 Script 8 — Relation Extraction

**Tujuan:** mengekstrak relasi sederhana berbasis pola kalimat dan menyimpan *evidence*.

Buat file:

```
scripts/08_relation_extraction.py

from pathlib import Path
import json
import re
import pandas as pd

CHUNK_FILE = Path("data/chunks/corpus_all.jsonl")
OUTPUT_FILE = Path("data/knowledge/relations.csv")
EVIDENCE_FILE =
Path("data/knowledge/evidence_table.csv")

RELATION_PATTERNS = [
 {
 "subject": "Mahasiswa",
 "relation": "mengisi",
 "object": "KRS",
 "terms": ["Mahasiswa", "KRS"],
 "verbs": ["mengisi", "wajib mengisi"],
 },
 {
 "subject": "Dosen Wali",
 "relation": "menyetujui",
 "object": "KRS",
 "terms": ["Dosen Wali", "KRS"],
 "verbs": ["menyetujui", "disetujui"],
 },
 {
 "subject": "KRS",
 "relation": "berisi",
 "object": "Mata Kuliah",
 "terms": ["KRS", "Mata Kuliah"],
 "verbs": ["berisi", "memuat"],
 },
 {
 "subject": "Mata Kuliah",
 "relation": "memiliki",
 "object": "SKS",
 "terms": ["Mata Kuliah", "SKS"],
 "verbs": ["memiliki", "berjumlah"],
 },
]
```

```

 },
]

def load_chunks():
 chunks = []

 with CHUNK_FILE.open("r", encoding="utf-8") as file:
 for line in file:
 line = line.strip()

 if line:
 chunks.append(json.loads(line))

 return chunks

def split_sentences(text):
 sentences = re.split(r"(?<=[.!?])\s+", text)
 return [sentence.strip() for sentence in sentences
 if sentence.strip()]

def contains_all_terms(sentence, terms):
 for term in terms:
 pattern = rf"\b{re.escape(term)}\b"

 if not re.search(pattern, sentence,
 flags=re.IGNORECASE):
 return False

 return True

def contains_any_verb(sentence, verbs):
 sentence_lower = sentence.lower()

 for verb in verbs:
 if verb.lower() in sentence_lower:
 return True

 return False

def main():
 OUTPUT_FILE.parent.mkdir(parents=True,
 exist_ok=True)

```

```

relations = []
evidence_rows = []
chunks = load_chunks()
evidence_counter = 1

for chunk in chunks:
 sentences = split_sentences(chunk["text"])

 for sentence in sentences:
 for pattern in RELATION_PATTERNS:
 if not contains_all_terms(sentence,
pattern["terms"]):
 continue

 if not contains_any_verb(sentence,
pattern["verbs"]):
 continue

 evidence_id =
f"EV{evidence_counter:04d}"

 row = {
 "subject": pattern["subject"],
 "relation": pattern["relation"],
 "object": pattern["object"],
 "evidence": sentence,
 "source_file": chunk["document_id"],
 "page": chunk.get("page_start", ""),
 "confidence": 0.90,
 "chunk_id": chunk["chunk_id"],
 "section": chunk["section"],
 "evidence_id": evidence_id,
 }

 relations.append(row)

 evidence_rows.append({
 "evidence_id": evidence_id,
 "evidence": sentence,
 "source_file": chunk["document_id"],
 "page": chunk.get("page_start", ""),
 "chunk_id": chunk["chunk_id"],
 "section": chunk["section"],
 })

 evidence_counter += 1

```

```

 relations_df =
pd.DataFrame(relations).drop_duplicates()
 evidence_df =
pd.DataFrame(evidence_rows).drop_duplicates()

 relations_df.to_csv(OUTPUT_FILE, index=False,
encoding="utf-8")
 evidence_df.to_csv(EVIDENCE_FILE, index=False,
encoding="utf-8")

 print(f"Relations dibuat: {OUTPUT_FILE}")
 print(f"Evidence table dibuat: {EVIDENCE_FILE}")
 print(f"Jumlah relasi: {len(relations_df)}")

if __name__ == "__main__":
 main()

```

Jalankan:

```
python scripts/08_relation_extraction.py
```

Output:

```

data/knowledge/relations.csv
data/knowledge/evidence_table.csv

```

Dalam dokumen terlampir, relasi harus memiliki *evidence*; tanpa bukti teks, hasilnya bukan *knowledge extraction*, tetapi dugaan.

## F.11 Script 9 — CSV ke RDF/Turtle

**Tujuan:** mengubah `relations.csv` menjadi RDF/Turtle.

Buat file:

```
scripts/09_csv_to_rdf_turtle.py

from pathlib import Path
import re
import pandas as pd
from rdflib import Graph, Namespace, Literal
from rdflib.namespace import RDF, RDFS, OWL

RELATIONS_FILE = Path("data/knowledge/relations.csv")
OUTPUT_TTL = Path("data/ontology/corpus_ontology.ttl")

EX = Namespace("http://example.org/corpus#")

def slugify(text):
 text = str(text).strip()
 text = text.replace(" ", "_")
 text = re.sub(r"^[A-Za-z0-9_]", "", text)
 return text

def main():
 if not RELATIONS_FILE.exists():
 raise FileNotFoundError(f"File tidak ditemukan: {RELATIONS_FILE}")

 OUTPUT_TTL.parent.mkdir(parents=True, exist_ok=True)

 df = pd.read_csv(RELATIONS_FILE)

 graph = Graph()
 graph.bind("ex", EX)
 graph.bind("rdf", RDF)
 graph.bind("rdfs", RDFS)
 graph.bind("owl", OWL)

 for _, row in df.iterrows():
 subject = str(row["subject"]).strip()
 relation = str(row["relation"]).strip()
 obj = str(row["object"]).strip()

 if not subject or not relation or not obj:
```

```

 continue

 subject_uri = EX[slugify(subject)]
 relation_uri = EX[slugify(relation)]
 object_uri = EX[slugify(obj)]

 graph.add((subject_uri, RDF.type,
OWL.NamedIndividual))
 graph.add((object_uri, RDF.type,
OWL.NamedIndividual))
 graph.add((relation_uri, RDF.type,
OWL.ObjectProperty))

 graph.add((subject_uri, RDFS.label,
Literal(subject, lang="id")))
 graph.add((object_uri, RDFS.label, Literal(obj,
lang="id")))
 graph.add((relation_uri, RDFS.label,
Literal(relation, lang="id")))

 graph.add((subject_uri, relation_uri,
object_uri))

 if "evidence" in row and
pd.notna(row["evidence"]):
 graph.add((subject_uri, EX.hasEvidence,
Literal(str(row["evidence"]), lang="id")))

 if "source_file" in row and
pd.notna(row["source_file"]):
 graph.add((subject_uri, EX.hasSourceFile,
Literal(str(row["source_file"]))))

 if "page" in row and pd.notna(row["page"]):
 graph.add((subject_uri, EX.hasPage,
Literal(str(row["page"]))))

 graph.serialize(destination=OUTPUT_TTL,
format="turtle")

 print(f"File RDF/Turtle dibuat: {OUTPUT_TTL}")
 print(f"Jumlah triple: {len(graph)}")

if __name__ == "__main__":
 main()

```

Jalankan:

```
python scripts/09_csv_to_rdf_turtle.py
```

Output:

```
data/ontology/corpus_ontology.ttl
```

Pendekatan ini sejalan dengan referensi *semantic knowledge modelling* yang menjelaskan konversi data tabular seperti CSV menjadi RDF/Turtle dengan membentuk RDF triples. RDF juga merepresentasikan data sebagai triple *subject–predicate–object*, sedangkan Turtle adalah salah satu format serialisasi yang lazim dipakai untuk data RDF.

## F.12 Script 10 — Jalankan Semua Pipeline

**Tujuan:** menjalankan semua script secara berurutan.

Buat file:

```
scripts/run_pipeline.py

import subprocess

SCRIPTS = [
 "scripts/01_check_pdf.py",
 "scripts/02_ocr_batch.py",
 "scripts/03_extract_text.py",
 "scripts/04_convert_markdown.py",
 "scripts/05_cleaning.py",
 "scripts/06_chunking.py",
 "scripts/07_entity_extraction.py",
 "scripts/08_relation_extraction.py",
 "scripts/09_csv_to_rdf_turtle.py",
]

def main():
 for script in SCRIPTS:
 print(f"\n=== Menjalankan {script} ===")
 subprocess.run(["python", script], check=True)

 print("\nPipeline selesai.")

if __name__ == "__main__":
 main()
```

Jalankan:

```
python scripts/run_pipeline.py
```

Output utama:

```
data/metadata/pdf_quality_report.csv
data/ocr/
data/raw_text/
data/markdown/
data/clean/
data/chunks/corpus_all.jsonl
data/metadata/chunk_report.csv
data/knowledge/entities.csv
data/knowledge/relations.csv
data/knowledge/evidence_table.csv
data/ontology/corpus_ontology.ttl
```

## F.13 Checklist Setelah Menjalankan Script

Pastikan file berikut tersedia:

```
data/metadata/pdf_quality_report.csv
data/clean/
data/chunks/corpus_all.jsonl
data/metadata/chunk_report.csv
data/knowledge/entities.csv
data/knowledge/relations.csv
data/knowledge/evidence_table.csv
data/ontology/corpus_ontology.ttl
```

Sebelum upload ke RAG, salin file final ke:

```
data/rag_upload/
├─ corpus_for_rag.md
├─ corpus_all.jsonl
└─ metadata.csv
```

## Catatan Penutup Lampiran F

Script di lampiran ini sengaja dibuat **ringkas, terbuka, dan mudah dimodifikasi mahasiswa**. Ia bukan pipeline industri yang sempurna, tetapi cukup untuk memahami proses inti: **PDF diaudit, di-OCR jika perlu, diekstrak, dibersihkan, dipotong menjadi chunk, diperkaya dengan entitas-relasi-evidence, lalu dikonversi ke RDF/Turtle**.

**Kalimat kunci Lampiran F:**

**Script hanyalah alat; kualitas corpus tetap ditentukan oleh audit, struktur, metadata, evidence, dan validasi manusia.**

# Glossary

Istilah	Definisi
<b>Annotation</b>	Penandaan tambahan pada teks, misalnya label entitas, konsep, relasi, atau kategori tertentu agar corpus lebih mudah dianalisis.
<b>Audit PDF</b>	Pemeriksaan awal terhadap PDF untuk mengetahui apakah dokumen <i>searchable</i> , scan, memiliki tabel, gambar, dua kolom, OCR buruk, atau perlu pemrosesan tambahan.
<b>Clean Corpus</b>	Corpus yang sudah dibersihkan dari noise, memiliki struktur, metadata, dan siap dipakai untuk <i>chunking</i> atau upload RAG.
<b>Cleaning</b>	Proses membersihkan teks dari header, footer, nomor halaman, watermark, spasi berlebihan, karakter OCR rusak, dan artefak ekstraksi.
<b>Chunk</b>	Potongan teks dari corpus yang dibuat agar mudah dicari oleh sistem RAG. Chunk harus cukup pendek untuk presisi, tetapi cukup panjang untuk menjaga konteks.
<b>Chunk ID</b>	Identitas unik untuk setiap chunk, misalnya <code>DOC001-0001</code> , agar potongan teks dapat dilacak kembali ke dokumen asal.
<b>Chunk Overlap</b>	Bagian teks yang sengaja diulang antar-chunk untuk menjaga konteks agar informasi penting tidak terputus.
<b>Chunk Report</b>	Laporan kualitas chunk, biasanya berisi <code>chunk_id</code> , panjang karakter, jumlah kata, section, dan catatan validasi.
<b>Concept</b>	Gagasan atau topik penting yang muncul dalam corpus, misalnya “Pengisian KRS”, “Cuti Akademik”, atau “Persetujuan Dosen Wali”.
<b>Confidence</b>	Nilai keyakinan awal terhadap hasil ekstraksi, misalnya dalam tabel relasi. Nilai rendah berarti perlu validasi manual.
<b>Corpus</b>	Kumpulan teks yang disusun, dibersihkan, diberi struktur, dan siap dipakai untuk NLP, RAG, pencarian, atau analisis pengetahuan.
<b>Corpus Index</b>	Daftar metadata seluruh dokumen dalam corpus, biasanya disimpan dalam <code>corpus_index.csv</code> .
<b>CSV</b>	Format tabel sederhana berbasis teks, sering dipakai untuk menyimpan metadata, daftar chunk, entitas, relasi, dan hasil evaluasi.
<b>Docling</b>	Tool <i>open source</i> untuk parsing dan konversi dokumen, termasuk PDF, ke format yang lebih siap untuk workflow AI seperti Markdown dan JSON.

<b>Document ID</b>	ID unik untuk dokumen, misalnya <b>DOC001</b> , agar semua output seperti metadata, chunk, entitas, dan relasi dapat dilacak ke dokumen asal.
<b>Document Type</b>	Jenis dokumen, misalnya panduan, SOP, brosur, regulasi, laporan, atau artikel.
<b>Embedding</b>	Representasi numerik dari teks yang digunakan oleh sistem RAG untuk mencari kemiripan makna antar teks.
<b>Entity</b>	Nama aktor, dokumen, proses, unit, istilah teknis, atau objek penting dalam corpus. Contoh: Mahasiswa, Dosen Wali, KRS, SKS.
<b>Entity Extraction</b>	Proses mengambil entitas penting dari teks menggunakan aturan, daftar istilah, NLP, atau bantuan LLM lokal.
<b>Evidence</b>	Bukti teks asli yang mendukung suatu konsep atau relasi. Dalam buku ini, setiap relasi penting harus memiliki evidence.
<b>Extraction Comparison</b>	Laporan perbandingan hasil ekstraksi dari beberapa tool, misalnya PyMuPDF, pdfplumber, OCRmyPDF, dan Docling.
<b>Extraction Tool</b>	Tool yang dipakai untuk mengambil isi PDF, misalnya PyMuPDF, pdfplumber, OCRmyPDF, atau Docling.
<b>Front Matter</b>	Blok metadata di awal file Markdown, biasanya ditulis dalam format YAML di antara tanda <code>---</code> .
<b>Git</b>	Tool <i>version control</i> untuk mencatat perubahan file corpus, script, metadata, dan dokumentasi proyek.
<b>Graph</b>	Struktur data berbentuk simpul dan relasi. Dalam konteks pengetahuan, graph dapat menunjukkan hubungan antar-entitas.
<b>Hallucination</b>	Jawaban atau relasi yang terdengar masuk akal tetapi tidak didukung oleh teks sumber. Dalam corpus RAG, halusinasi dicegah dengan metadata dan evidence.
<b>Heading</b>	Judul bagian dalam dokumen, misalnya <b># Bab 1</b> atau <b>## Pengisian KRS</b> . Heading penting untuk chunking berbasis struktur.
<b>JSONL</b>	Format satu baris satu objek JSON. Cocok untuk menyimpan chunk karena setiap baris dapat berisi <b>chunk_id</b> , <b>document_id</b> , <b>section</b> , dan <b>text</b> .
<b>Knowledge Base</b>	Tempat penyimpanan pengetahuan yang dapat dicari oleh sistem RAG atau LLM. Dalam konteks buku ini, isinya berasal dari corpus yang sudah disiapkan.
<b>Knowledge Extraction</b>	Proses mengambil entitas, konsep, relasi, dan evidence dari corpus.

<b>Knowledge Graph</b>	Representasi pengetahuan sebagai jaringan entitas dan relasi, misalnya <i>Mahasiswa</i> → <i>mengisi</i> → <i>KRS</i> .
<b>Knowledge Layer</b>	Lapisan tambahan di atas corpus berupa entitas, konsep, relasi, dan evidence agar corpus lebih mudah diuji dan dikembangkan.
<b>Language</b>	Bahasa dokumen, misalnya <i>id</i> untuk Indonesia atau <i>en</i> untuk Inggris. Metadata bahasa penting untuk OCR dan pemrosesan teks.
<b>LLM</b>	<i>Large Language Model</i> , model bahasa besar yang dapat menghasilkan jawaban berdasarkan konteks yang diberikan.
<b>Markdown</b>	Format teks ringan yang mudah dibaca manusia dan mudah diproses mesin. Cocok untuk menyimpan clean corpus.
<b>Metadata</b>	Informasi tentang dokumen atau chunk, seperti judul, sumber file, halaman, bahasa, jenis dokumen, tool ekstraksi, dan status.
<b>Metadata CSV</b>	File CSV yang menyimpan metadata dokumen atau corpus, misalnya <i>metadata.csv</i> atau <i>corpus_index.csv</i> .
<b>Named Individual</b>	Individu spesifik dalam ontologi OWL, misalnya Mahasiswa, Dosen Wali, atau KRS sebagai objek yang diberi identitas.
<b>Noise</b>	Gangguan dalam teks seperti header-footer berulang, watermark, nomor halaman, karakter OCR rusak, atau teks tabel yang kacau.
<b>Normalization</b>	Proses menyeragamkan teks, termasuk spasi, tanda baca, kapitalisasi, paragraf, dan istilah.
<b>OCR</b>	<i>Optical Character Recognition</i> , proses mengubah gambar teks dari PDF scan menjadi teks yang dapat dicari.
<b>OCR Noise</b>	Kesalahan hasil OCR, misalnya huruf salah terbaca, kata terpecah, angka berubah, atau istilah teknis rusak.
<b>OCRmyPDF</b>	Tool <i>open source</i> untuk menambahkan lapisan teks OCR pada PDF scan sehingga dokumen dapat dicari.
<b>Ontology</b>	Struktur pengetahuan yang mendefinisikan konsep, kelas, individu, relasi, dan properti. Dalam buku ini dipakai secara ringan untuk memperkaya corpus RAG.
<b>Ontology Ringan</b>	Ontologi sederhana yang berisi peta konsep dan relasi utama, tanpa membebani mahasiswa dengan logika formal yang rumit.
<b>OWL</b>	<i>Web Ontology Language</i> , bahasa standar untuk membuat ontologi yang lebih kaya daripada RDF biasa.
<b>Page End</b>	Halaman akhir sumber dokumen atau chunk. Berguna untuk pelacakan sumber.

<b>Page Start</b>	Halaman awal sumber dokumen atau chunk. Berguna untuk audit dan sitasi.
<b>Parsing</b>	Proses membaca dan memecah dokumen menjadi struktur yang lebih dapat diproses, seperti teks, tabel, heading, dan paragraf.
<b>PDF Searchable</b>	PDF yang memiliki lapisan teks sehingga isinya dapat dicari, disalin, dan diekstrak.
<b>PDF Scan</b>	PDF yang berisi gambar halaman, bukan teks langsung. Biasanya membutuhkan OCR.
<b>pdfplumber</b>	Library Python untuk mengekstrak teks, tabel, dan elemen layout dari PDF berbasis teks.
<b>Pipeline</b>	Rangkaian proses kerja dari PDF mentah sampai corpus siap upload ke RAG.
<b>Prompt</b>	Instruksi yang diberikan kepada LLM untuk melakukan tugas tertentu, misalnya ekstraksi konsep, validasi relasi, atau evaluasi jawaban.
<b>Protégé</b>	Tool <i>open source</i> untuk membuat, melihat, dan mengedit ontologi berbasis OWL.
<b>PyMuPDF</b>	Library Python untuk membaca PDF, mengekstrak teks, dan mengambil informasi halaman.
<b>RAG</b>	<i>Retrieval-Augmented Generation</i> , pendekatan di mana sistem mencari dokumen relevan terlebih dahulu, lalu LLM membuat jawaban berdasarkan hasil pencarian.
<b>RDF</b>	<i>Resource Description Framework</i> , model data berbasis triple <b>subject–predicate–object</b> untuk merepresentasikan pengetahuan sebagai graph.
<b>RDF Triple</b>	Struktur pengetahuan berbentuk <b>subject → predicate → object</b> , misalnya <b>Mahasiswa → mengisi → KRS</b> .
<b>rdflib</b>	Library Python untuk membuat, membaca, dan menyimpan RDF, termasuk format Turtle.
<b>Relation</b>	Hubungan antara dua entitas atau konsep, misalnya <b>Mahasiswa mengisi KRS</b> .
<b>Relation Extraction</b>	Proses mengambil relasi dari teks, baik dengan pola kalimat, aturan sederhana, NLP, maupun bantuan LLM lokal.
<b>Retrieval</b>	Proses pencarian chunk atau dokumen yang relevan terhadap pertanyaan pengguna.

<b>Section</b>	Bagian dokumen tempat chunk berasal, biasanya mengikuti heading seperti “Pengisian KRS” atau “Cutu Akademik”.
<b>Semantic Search</b>	Pencarian berdasarkan kemiripan makna, bukan hanya kecocokan kata persis.
<b>Source File</b>	Nama file PDF asal dari sebuah corpus, chunk, entitas, atau relasi.
<b>SPARQL</b>	Bahasa query untuk mengambil informasi dari data RDF atau graph pengetahuan.
<b>spaCy</b>	Library NLP <i>open source</i> untuk tokenisasi, entitas, aturan pola, dan pemrosesan bahasa.
<b>Stanza</b>	Library NLP <i>open source</i> untuk pemrosesan bahasa, termasuk tokenisasi, parsing, dan pengenalan entitas.
<b>Status</b>	Penanda tahap dokumen atau corpus, misalnya <code>raw</code> , <code>ocr_done</code> , <code>clean</code> , <code>chunked</code> , atau <code>ready_for_rag</code> .
<b>Subject</b>	Bagian pertama dalam relasi atau RDF triple. Contoh: <code>Mahasiswa</code> dalam <code>Mahasiswa → mengisi → KRS</code> .
<b>Tesseract</b>	Mesin OCR <i>open source</i> yang dapat digunakan untuk mengenali teks dalam gambar atau PDF scan.
<b>Text Extraction</b>	Proses mengambil teks dari PDF atau dokumen lain agar dapat dibersihkan dan diproses.
<b>Token</b>	Unit kecil dalam teks, bisa berupa kata, tanda baca, atau potongan kata, yang diproses oleh sistem NLP atau LLM.
<b>Traceability</b>	Kemampuan melacak hasil jawaban, chunk, relasi, atau evidence kembali ke dokumen dan halaman sumber.
<b>Turtle</b>	Format serialisasi RDF yang lebih ringkas dan mudah dibaca manusia, biasanya berekstensi <code>.ttl</code> .
<b>Upload RAG</b>	Tahap memasukkan file final seperti Markdown, JSONL, atau metadata ke sistem RAG atau Knowledge Base.
<b>Vector Database</b>	Basis data yang menyimpan embedding teks untuk pencarian berbasis kemiripan makna.
<b>Version Control</b>	Sistem untuk mencatat perubahan file proyek dari waktu ke waktu, biasanya menggunakan Git.
<b>YAML</b>	Format teks sederhana untuk menyimpan metadata terstruktur, sering dipakai sebagai <i>front matter</i> dalam Markdown.

# Referensi

Bock, J. (2013). *Ontology alignment using biologically-inspired optimisation algorithms*. KIT Scientific Publishing. ISBN 978-3-86644-936-7. <http://www.ksp.kit.edu>

Sumber attached menunjukkan judul, penulis, KIT Scientific Publishing, tahun 2013, lisensi, dan ISBN.

CommonMark. (n.d.). *CommonMark: A strongly defined, highly compatible specification of Markdown*. <https://commonmark.org/>

Docling Project. (n.d.). *Docling documentation*. <https://docling-project.github.io/docling/>  
Dokumentasi resmi menyebut Docling mendukung parsing banyak format, termasuk PDF, serta ekspor ke Markdown dan JSON untuk workflow AI/RAG. ([docling-project.github.io](https://docling-project.github.io))

Docling Project. (n.d.). *Docling: Get your documents ready for gen AI*. GitHub.  
<https://github.com/docling-project/docling>

Repositori resmi menjelaskan Docling sebagai tool pemrosesan dokumen dengan kemampuan *advanced PDF understanding* dan integrasi ke ekosistem *generative AI*. ([GitHub](#))

Git. (n.d.). *Git documentation*. <https://git-scm.com/doc>

JSON Lines. (n.d.). *JSON Lines*. <https://jsonlines.org/>

Kaswan, K. S., Dhatteerwal, J. S., & Nayyar, A. (2025). *Digital personality: A man forever: Volume 3: Ontologies to dialogue generation*. CRC Press.  
<https://doi.org/10.1201/9781003530701>

Sumber attached mencatat edisi pertama 2025, CRC Press, ISBN cetak dan digital, serta DOI 10.1201/9781003530701.

Keet, C. M. (2020). *An introduction to ontology engineering* (Version 1.5). University of Cape Town. <https://people.cs.uct.ac.za/~mkeet/OEbook/>

Sumber attached mencantumkan judul, penulis, versi 1.5 tahun 2020, lisensi CC BY 4.0, dan situs resmi textbook.

LangChain. (n.d.). *Docling integration*.

[https://docs.langchain.com/oss/python/integrations/document\\_loaders/docling](https://docs.langchain.com/oss/python/integrations/document_loaders/docling)

Dokumentasi ini menjelaskan bahwa Docling dapat mem-parsing PDF dan format dokumen lain ke representasi yang siap untuk workflow *generative AI* seperti RAG. ([Dokumen LangChain](#))

LangChain. (n.d.). *Recursive text splitter*.

[https://docs.langchain.com/oss/python/integrations/splitters/recursive\\_text\\_splitter](https://docs.langchain.com/oss/python/integrations/splitters/recursive_text_splitter)

OCRmyPDF. (n.d.). *OCRmyPDF documentation*. <https://ocrmypdf.readthedocs.io/>

Dokumentasi resmi menjelaskan bahwa OCRmyPDF menambahkan lapisan teks OCR ke PDF scan agar dapat dicari. ([ocrmypdf.readthedocs.io](https://ocrmypdf.readthedocs.io))

OCRmyPDF. (n.d.). *OCRmyPDF GitHub repository*. <https://github.com/ocrmypdf/ocrmypdf>  
Repositori resmi mencatat fitur seperti membuat PDF/A searchable, menempatkan teks OCR di bawah gambar, menjaga resolusi gambar, *deskew*, *clean*, dan validasi file input-output. ([GitHub](#))

Oltramari, A., Vossen, P., Qin, L., & Hovy, E. (Eds.). (2013). *New trends of research in ontologies and lexical resources: Ideas, projects, systems*. Springer.

<https://doi.org/10.1007/978-3-642-31782-8>

Sumber attached dan Springer mencatat editor, judul, seri *Theory and Applications of Natural Language Processing*, penerbit Springer, ISBN, dan DOI. ([Springer Nature Link](#))

Open WebUI. (n.d.). *Knowledge*.

<https://docs.openwebui.com/features/workspace/knowledge/>

Dokumentasi resmi menjelaskan *Knowledge Base*, upload dokumen, *Focused Retrieval / RAG*, *Full Context*, dan pencarian dokumen berbasis makna. ([Open WebUI](#))

Open WebUI. (n.d.). *Retrieval augmented generation (RAG)*.

<https://docs.openwebui.com/features/chat-conversations/rag/>

Dokumentasi ini menjelaskan konfigurasi RAG, termasuk *Markdown Header Splitting* untuk menjaga struktur dokumen saat pemecahan chunk. ([Open WebUI](#))

pdfplumber. (n.d.). *pdfplumber*. GitHub. <https://github.com/jsvine/pdfplumber>

Python Software Foundation. (n.d.). *re — Regular expression operations*.

<https://docs.python.org/3/library/re.html>

Python Software Foundation. (n.d.). *venv — Creation of virtual environments*.

<https://docs.python.org/3/library/venv.html>

PyMuPDF. (n.d.). *PyMuPDF documentation: Text*.

<https://pymupdf.readthedocs.io/en/latest/recipes-text.html>

Dokumentasi PyMuPDF menjelaskan ekstraksi teks memakai `Page.get_text()`, termasuk teks, blok, kata, dan informasi posisi. ([pymupdf.readthedocs.io](https://pymupdf.readthedocs.io))

PyMuPDF. (n.d.). *PyMuPDF documentation: The basics*.

<https://pymupdf.readthedocs.io/en/latest/the-basics.html>

Dokumentasi dasar PyMuPDF juga mencatat kemampuan ekstraksi teks ke Markdown. ([pymupdf.readthedocs.io](https://pymupdf.readthedocs.io))

RDFLib. (n.d.). *RDFLib documentation*. <https://rdflib.readthedocs.io/>

Dokumentasi RDFLib menyebut library ini sebagai paket Python untuk bekerja dengan RDF, termasuk parser/serializer RDF/XML, Turtle, JSON-LD, dan dukungan SPARQL 1.1.

([rdflib.readthedocs.io](https://rdflib.readthedocs.io))

spaCy. (n.d.). *Rule-based matching*. <https://spacy.io/usage/rule-based-matching>

Stanza. (n.d.). *Stanza: A Python NLP package for many human languages*.

<https://stanfordnlp.github.io/stanza/>

Tesseract OCR. (n.d.). *Tesseract documentation*. <https://tesseract-ocr.github.io/>

Theocharis, S., & Tsihrintzis, G. A. (2023). *Semantic knowledge modelling via open linked ontologies: Ontologies in e-governance*. Springer Nature Switzerland.  
<https://doi.org/10.1007/978-3-031-20585-9>

Sumber attached mencantumkan judul, penulis, Springer Nature Switzerland, ISBN, DOI, dan fokus pada Semantic Web, linked open data, RDF, OWL, SPARQL, dan knowledge modelling.

World Wide Web Consortium. (2012). *OWL 2 Web Ontology Language primer* (2nd ed.). W3C Recommendation. <https://www.w3.org/TR/owl2-primer/>  
W3C menjelaskan OWL 2 memakai *individuals*, *classes*, dan *properties* untuk merepresentasikan objek, kategori, dan relasi. (W3C)

World Wide Web Consortium. (2013). *SPARQL 1.1 overview*. W3C Recommendation. <https://www.w3.org/TR/sparql11-overview/>  
SPARQL 1.1 didefinisikan sebagai spesifikasi untuk query dan manipulasi konten RDF graph di Web atau RDF store. (W3C)

World Wide Web Consortium. (2014a). *RDF 1.1 concepts and abstract syntax*. W3C Recommendation. <https://www.w3.org/TR/rdf11-concepts/>  
W3C menjelaskan RDF triple sebagai relasi yang menghubungkan *subject* dan *object* melalui *predicate*. (W3C)

World Wide Web Consortium. (2014b). *RDF 1.1 primer*. W3C Working Group Note. <https://www.w3.org/TR/rdf11-primer/>  
Primer ini menjelaskan RDF statement sebagai hubungan antara dua resource, dengan *subject*, *predicate*, dan *object* sebagai triple. (W3C)

World Wide Web Consortium. (2014c). *RDF Schema 1.1*. W3C Recommendation. <https://www.w3.org/TR/rdf-schema/>

## Tentang Penulis



**Onno W. Purbo**, saat ini bertugas sebagai rektor di Institut Teknologi Tangerang Selatan (ITTS). Onno memperoleh gelar Ph.D bidang Electrical Engineering dari University of Waterloo, Canada, adalah seorang copyleft, educator dan ICT evangelist. Dia sudah mempublikasikan 50++ buku, termasuk free ICT ebook untuk sekolah tahun 2008. Beberapa buku terakhirnya adalah "Internet-TCP/IP: Konsep Dan Implementasi", 2018; "Sistem Operasi, Konsep Dan Membuat Linux OpenWRT Dan ROM Android", 2019; "IPv6 Untuk Mendukung Operasi Jaringan Dan Domain Name System", 2019; "Kubernetes untuk Pemula", 2024 dan "Membuat Operator Seluler 5G Sendiri", 2024.

Dia memimpin sambungan pertama Internet di Institut Teknologi Bandung, tahun 1993-2000, dan menggunakannya untuk membuat jaringan Internet pendidikan yang pertama di Indonesia. Dia membebaskan frekuensi WiFi, memperkenalkan RT/RW-net, antenna Wajanbolic dan jaringan selular OpenBTS dan private 5G sendiri. Dia memimpin jaringan telepon pertama di atas Internet, VoIP Merdeka, yang kemudian hari dikenal sebagai VoIP Rakyat berbasis SIP dan menggunakan kode area +62520 dan +62521. Dia saat ini aktif memperkenalkan e-Learning, dan menjalankan server e-Learning di <http://lms.onnocenter.or.id/moodle/> 66,000++ siswa / mahasiswa dan <https://opencourse.itts.ac.id> 39.000+ mahasiswa secara gratis. Email: [onno@indo.net.id](mailto:onno@indo.net.id)