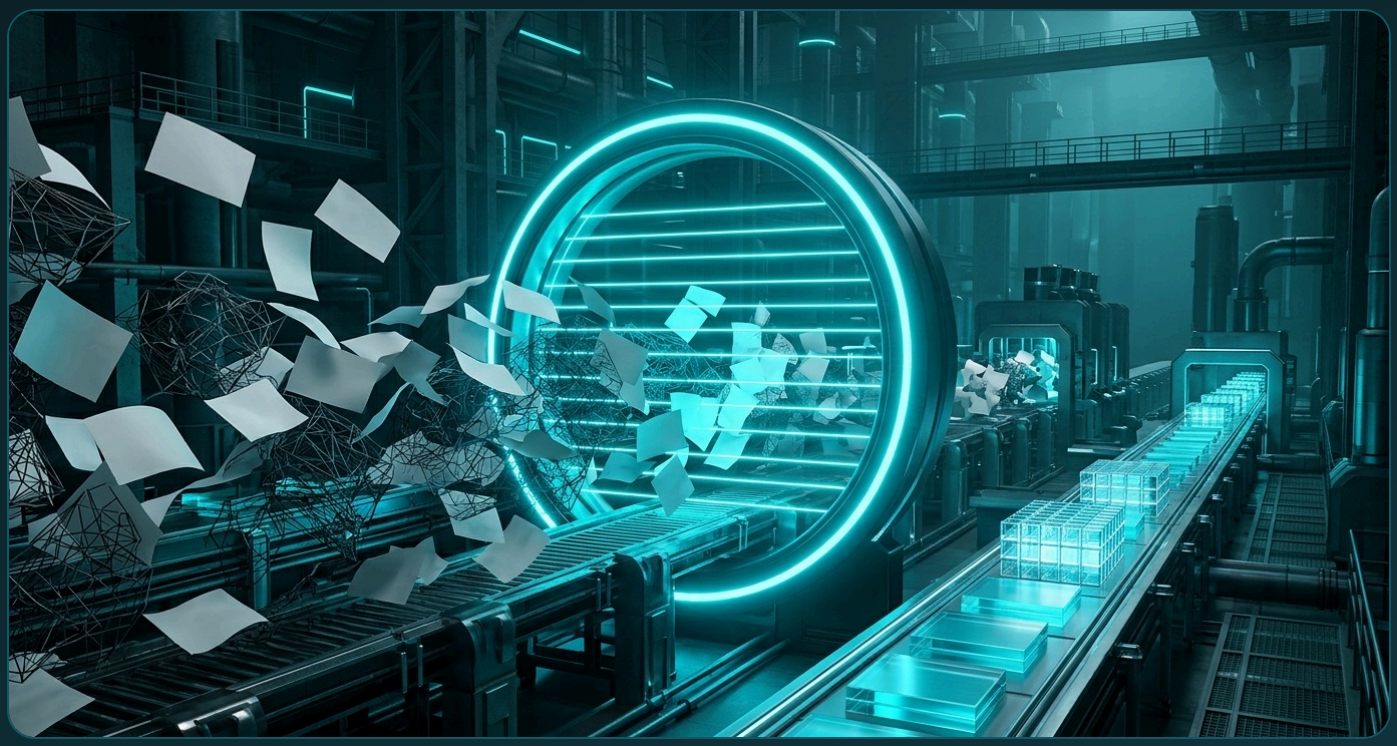
AI AGENT ENGINEERING & ARCHITECTURE • FIELD GUIDE

AI Document & Web Scraping Pipelines

The complete 12-page visual handbook to transforming messy PDFs, Office suites, and JavaScript-heavy websites into clean, chunked, citation-ready knowledge with Docling, Crawl4AI, and Firecrawl.

DoclingDocument & TableFormer | Granite-Docling VLM & OCR | Crawl4AI Async & CSS Schemas | fit_markdown Noise Pruning | Firecrawl v2 /extract & Actions | Sanitization & SHA-256 Dedup



Structure-First Parsing
Preserve reading order, merged table headers, formulas, and page provenance instead of flattening layouts into broken plain text.

Zero-Bloat Web Markdown
Strip navigation chrome, ads, and footers using PruningContentFilter and declarative schemas before tokens touch your LLM.

Hardened Ingestion Hygiene
Quarantine hidden prompt-injection lures, honor robots.txt crawl delays, and skip redundant embeds via SHA-256 hashing.

The Ingestion Boundary: Raw Files & Live Web to Clean Context

Retrieval systems and autonomous agents rarely fail because the language model cannot reason; they fail because noisy ingestion feeds scrambled tables, broken multi-column reading order, and sprawling navigation bars into the context window.

Quick Navigation

1. Core Mental Model & Ingestion Architecture p. 2
2. Setup, Hardware Tiers & Engine Matrix p. 3
3. Annotated Ingestion & Chunk Inspector Tour p. 4
4. Docling: PDF Layout, TableFormer & OCR p. 5
5. Crawl4AI: Async Crawling & fit_markdown p. 6
6. Firecrawl v2: /map, /extract & Webhooks p. 7
7. HybridChunker, Provenance & Metadata p. 8
8. Five Copy-Paste Production Recipes p. 9
9. Crawl Ethics, Quarantine & Deduplication p. 10
10. End-to-End Cross-Tool Synergy Pipeline p. 11
11. Troubleshooting Clinic & Pre-Flight Checklist p. 12

D Docling (Files)

Local-first parser for PDFs, DOCX, PPTX, and scanned images. Builds a unified `DoclingDocument` hierarchy with TableFormer cell geometry.

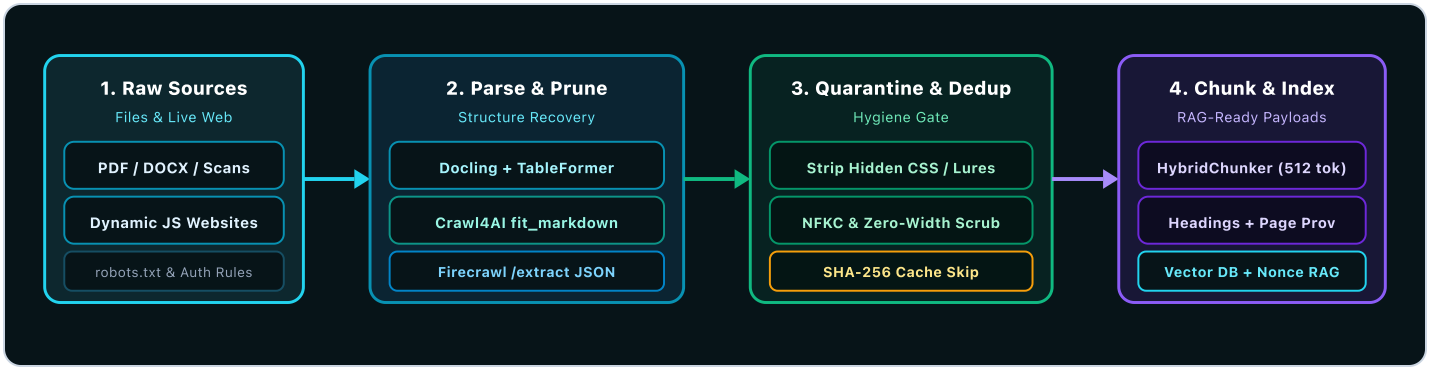
C Crawl4AI (Local Web)

Open-source async Playwright crawler for dynamic SPAs. Produces low-token `fit_markdown` and zero-LLM CSS or XPath JSON payloads.

F Firecrawl v2 (API)

Cloud or self-hosted API offering instant `/map` discovery, schema-guided `/extract`, browser actions, and persistent change tracking.

Architecture: Raw Sources → Specialized Parsers → Sanitize & Dedup → Provenance Chunks



Why Unified Normalization Pays Off Immediately

When local PDF parsers and live web scrapers output different ad-hoc formats, downstream agents choke on edge cases. Routing every source into one normalized contract—pruned Markdown, strict JSON records, and chunk provenance metadata—lets your RAG index and tool loops operate cleanly regardless of where knowledge originated.

Setup, Hardware Tiers & Extraction Engine Matrix

No single scraper handles every source well. Match PDFs, Office documents, open web crawls, and protected portals to the right local library or managed API before writing pipeline code.

Engine & Mode	Best Input Sources	Compute & Cost	Primary Outputs	Strengths & Tradeoffs
Docling Standard Layout + TableFormer	Native PDFs, DOCX, PPTX, XLSX, HTML, audio	Local CPU or GPU; zero API fees	`DoclingDocument`, Markdown, lossless JSON	Preserves reading order and merged table spans cleanly on local hardware.
Docling VLM Granite-Docling 258M	Complex math, dense scans, chart-heavy PDFs	Apple MPS or NVIDIA GPU recommended	Compact DocTags, Markdown, HTML	End-to-end vision parsing replaces multi-step OCR chaining; heavier VRAM footprint.
Crawl4AI (CSS) JsonCssExtraction	Repeated catalog cards, docs, tables, SPAs	Local Playwright; zero LLM token cost	Structured JSON + `fit_markdown`	Blazing fast and deterministic once CSS or XPath selectors are mapped.
Crawl4AI (LLM) LLMExtractionStrategy	Irregular layouts, articles, mixed prose pages	Local Playwright + LLM API tokens	Pydantic JSON + pruned Markdown	Handles changing DOM shapes flexibly; costs tokens per page unless pruned first.
Firecrawl v2 Cloud API + firecrawl-mcp	Hard-to-scrape sites, sitemaps, live monitors	Managed SaaS credits; 2-day cache saves hits	Markdown, v2 JSON, screenshots, diffs	Built-in proxies, anti-bot retries, browser actions, and zero infra setup.
Firecrawl Self-Host Docker Compose	Internal intranets, private VPC knowledge	Node.js + Redis + Playwright (2 GB+ RAM)	Universal REST API across languages	Full data sovereignty; requires maintaining your own proxy pools and containers.

Quick Installation & Runtime Sizing

1 Local Python Stack (Docling + Crawl4AI)

Install both open-source toolkits in a Python 3.10+ virtual environment and initialize Playwright browser binaries in one step:

```
pip install docling crawl4ai pydantic
crawl4ai-setup
```

Sizing tip: Standard Docling runs comfortably on 4–8 GB RAM; Apple Silicon MPS or CUDA GPUs cut `Granite-Docling` and `TableFormer` page latency by 4x–8x.

2 Firecrawl v2 SDK & MCP Server

Connect agents or batch scripts to Firecrawl using the official v2 Python SDK or drop-in Model Context Protocol server:

```
pip install firecrawl
FIRECRAWL_API_KEY=fc-... npx -y firecrawl-mcp
```

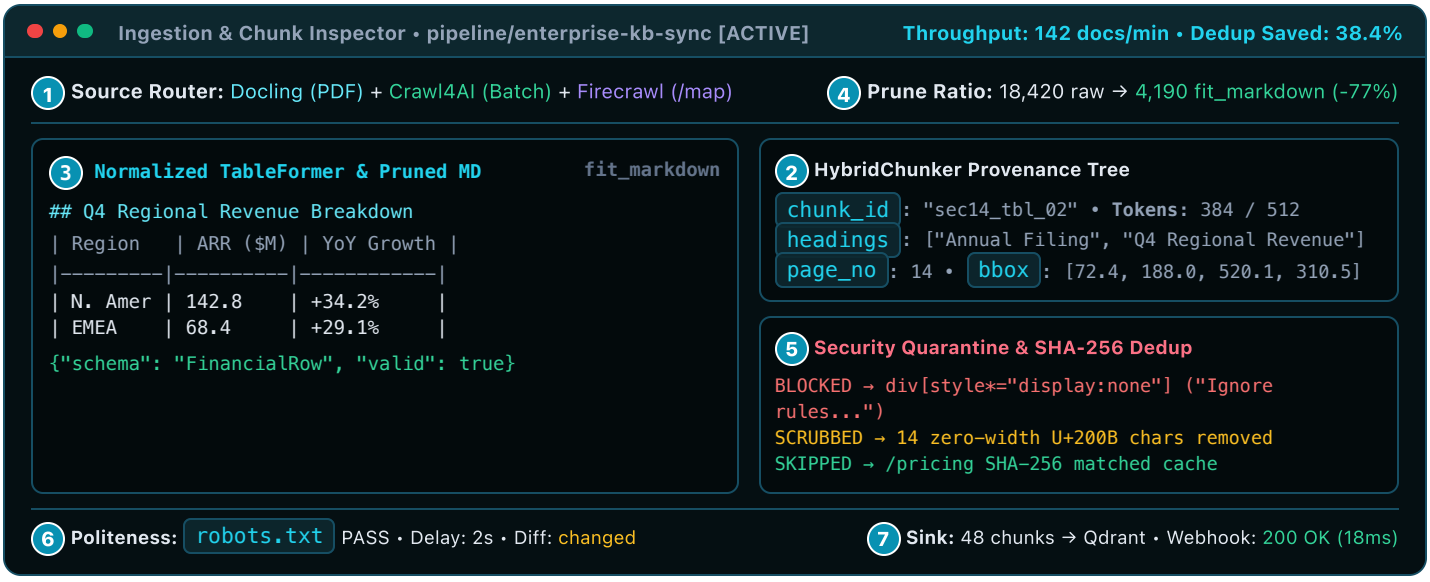
Quota tip: Firecrawl v2 caches scrapes for 2 days by default (`maxAge`), returning cached semantic index snapshots without burning fresh browser time.

Avoid the Naive HTTP + BeautifulSoup Trap

Fetching URLs with plain HTTP clients and flattening HTML with `soup.get_text()` fails silently on modern React and Next.js portals, merges sidebar links into paragraphs, and destroys financial PDF columns. Starting with layout-aware document parsers and pruned browser crawlers saves weeks of downstream prompt debugging.

Annotated Workspace: Ingestion & Chunk Inspector HUD

Reliable data engineering pipelines expose live observability across every stage: source routing, token pruning ratios, prompt-injection drops, SHA-256 deduplication, and exact chunk provenance.



Control Legend & Inspection Points

1 **Multi-Engine Source Router:** Dispatches binary documents to Docling, open batch web lists to Crawl4AI, and guarded or monitored portals to Firecrawl.

2 **Provenance & Chunk Tree:** Inspects token counts alongside `headings` breadcrumbs, `page_no`, and layout bounding boxes before embedding.

3 **Normalized Output Inspector:** Compares TableFormer pipe tables, `fit_markdown` prose, and schema-validated JSON records side by side.

4 **Boilerplate Compression Gauge:** Measures raw HTML versus pruned Markdown bytes so navigation bars and footer links never inflate token bills.

5 **Quarantine & Dedup Ledger:** Flags stripped invisible CSS traps, scrubbed Unicode smuggling bytes, and unchanged pages skipped via `SHA-256` hashing.

6 **Politeness & Change Monitor:** Tracks `robots.txt` rules, adaptive backoff delays, and Firecrawl `changeTracking` diffs across scheduled runs.

7 **Downstream Sink & Webhooks:** Confirms atomic vector database upserts and triggers real-time webhook notifications on document updates.

Spotting Silent Ingestion Degradation Before Users Notice

Always log the raw-to-fit compression ratio and extracted chunk count per URL pattern. A sudden drop in `fit_markdown` length or a spike in empty chunks usually means a target site shipped a new JavaScript paywall, cookie modal, or CSS class change that requires updating your `wait_for` selector.

Docling: PDF Layout, TableFormer & VLM Conversion

Developed by IBM Research, Docling replaces brittle text dumpers with deep layout analysis, converting PDFs, Word files, slide decks, and scanned archives into a structured `DoclingDocument` tree.

1 Unified `DoclingDocument`

Represents titles, nested headings, reading order, captions, code blocks, and formulas in one schema exporting cleanly to Markdown, HTML, JSON, or DocTags.

2 `TableFormer` Cell Grid

Predicts bounding boxes, row spans, and multi-tier headers. Choose `FAST` for high volume or `ACCURATE` for dense financial tables.

3 Modular OCR & VLM

Swap between `RapidOCR`, `EasyOCR`, `Tesseract`, and macOS Vision, or run `Granite-Docling` (258M) end-to-end via `VlmPipeline`.

Docling Formula: `Pipeline Options` (`PdfPipelineOptions`) + `Selective OCR` (`RapidOcrOptions`) + `Table Mode` (`TableFormerMode.ACCURATE`) + `Converter` (`DocumentConverter`)

```
from docling.datamodel.base_models import InputFormat
from docling.datamodel.pipeline_options import PdfPipelineOptions, RapidOcrOptions, TableFormerMode
from docling.document_converter import DocumentConverter, PdfFormatOption

opts = PdfPipelineOptions(do_ocr=True, do_table_structure=True)
opts.ocr_options = RapidOcrOptions(force_full_page_ocr=False) # Run OCR only on scanned regions
opts.table_structure_options.mode = TableFormerMode.ACCURATE  # Best fidelity for merged headers

converter = DocumentConverter(format_options={InputFormat.PDF: PdfFormatOption(pipeline_options=opts)})
doc = converter.convert("quarterly_report.pdf").document
print(doc.export_to_markdown()) # CLI VLM option: docling --pipeline vlm --vlm-model granite_docling file.pdf
```



Figure 5.1 • Structural Layout Decomposition: Docling separates multi-column narrative text, nested TableFormer cell grids, and embedded visual figures into distinct typed nodes before serializing reading-order Markdown.

Crawl4AI: Async Playwright, Pruning & Schema Extraction

Crawl4AI gives Python engineers direct programmatic control over headless Chromium instances—pairing concurrent crawling with heuristic Markdown noise pruning and deterministic CSS extraction schemas.

1 Zero-LLM `JsonCssExtractionStrategy`

Map repeating DOM cards once using CSS or XPath selectors. Extracting 10,000 uniform catalog or directory pages costs zero LLM tokens and finishes deterministically in milliseconds.

2 Schema-Guided `LLMExtractionStrategy`

When page HTML varies wildly, supply a Pydantic schema via `LLMConfig` and set `input_format="fit_markdown"` so the model only reads high-signal body text.

Crawl4AI Formula: `Browser Env` (`BrowserConfig`) + `Noise Pruner` (`PruningContentFilter`) + `CSS / LLM Extract` (`extraction_strategy`) + `Wait & Session` (`wait_for` , `session_id`)

```
import asyncio, json
from crawl4ai import AsyncWebCrawler, BrowserConfig, CrawlerRunConfig, CacheMode
from crawl4ai.content_filter_strategy import PruningContentFilter
from crawl4ai.markdown_generation_strategy import DefaultMarkdownGenerator
from crawl4ai.extraction_strategy import JsonCssExtractionStrategy

async def scrape_catalog():
    md_gen = DefaultMarkdownGenerator(content_filter=PruningContentFilter(threshold=0.4, min_word_threshold=20))
    schema = {"name": "Posts", "baseSelector": "article.post", "fields": [
        {"name": "title", "selector": "h2", "type": "text"},
        {"name": "href", "selector": "a", "type": "attribute", "attribute": "href"},
    ]}
    run_cfg = CrawlerRunConfig(cache_mode=CacheMode.BYPASS, markdown_generator=md_gen,
        extraction_strategy=JsonCssExtractionStrategy(schema), wait_for="css:article.post")
    async with AsyncWebCrawler(config=BrowserConfig(headless=True)) as crawler:
        res = await crawler.arun(url="https://example.com/blog", config=run_cfg)
        return res.markdown.fit_markdown, json.loads(res.extracted_content or "[]")
```

P `PruningContentFilter` vs. `BM25`

`PruningContentFilter` scores DOM link density and text length to strip menus and footers across entire sites. Use `BM25ContentFilter(user_query=...)` when you only want blocks matching a specific search topic.

S Stateful `session_id` & Lifecycle Hooks

Keep a single browser tab alive across pagination clicks by reusing `session_id` with `js_only=True` and `js_code`, and inject cookies or auth headers safely inside `on_page_context_created`.

Always Attach a Filter Before Reading `fit_markdown`

A frequent beginner pitfall is inspecting `result.markdown.fit_markdown` without passing a content filter to `DefaultMarkdownGenerator`—which leaves `fit_markdown` empty. Explicitly attach `PruningContentFilter` or `BM25ContentFilter` to unlock 60–80% boilerplate token reduction.

Firecrawl v2: Mapping, Structured `/extract` & Change Tracking

Firecrawl v2 provides managed browser infrastructure, fast sitemap URL discovery, schema-guided JSON extraction, interactive page actions, and persistent change diffs over a clean REST and Python SDK.



Figure 7.1 • Managed Extraction & Snapshot Vault: Firecrawl filters boilerplate noise and blocked markup at the browser edge, streaming schema-validated JSON records and persistent change-tracking diffs into your downstream store.

1 Fast `/map` & Smart `/crawl`

Inventory thousands of URLs via `firecrawl.map()` before scraping, then scope recursive crawls with `includePaths` or natural-language prompts previewed via `/crawl-params-preview`.

2 JSON Mode, `/extract` & Actions

Pass `{"type": "json", "schema": ...}` inside `formats` for single URLs (or `/extract` for wildcard domains), paired with pre-scrape `actions` like `click`, `write`, and `wait`.

3 `changeTracking` & Webhooks

Compare live pages against persistent snapshots using `git-diff` or `json` modes to get `new`, `same`, `changed`, or `removed` events via HTTPS webhooks.

```
from firecrawl import Firecrawl
from pydantic import BaseModel

class PricingTier(BaseModel):
    plan_name: str
    monthly_usd: float

fc = Firecrawl(api_key="fc-YOUR_API_KEY") # Or run keyless sandbox / self-hosted endpoint
urls = fc.map(url="https://docs.example.com", limit=100, sitemap="include")

doc = fc.scrape(
    "https://example.com/pricing",
    actions=[{"type": "click", "selector": "#annual-toggle"}, {"type": "wait", "milliseconds": 600}],
    formats=["markdown", {"type": "json", "schema": PricingTier.model_json_schema()},
             {"type": "changeTracking", "modes": ["git-diff", "json"]}],
)
print(doc.change_tracking.get("changeStatus"), doc.json) # Webhooks fire on started, page, completed
```

HybridChunker, Contextualization & Provenance Metadata

Clean Markdown is only half the ingestion pipeline. Naive character splitters tear apart Markdown tables and strip parent headings; structure-aware chunking preserves semantics and clickable citations.

Provenance Formula: `Token Cap` (`max_tokens=512`) + `Heading Path` (`meta.headings`) + `Table Repeat` (`repeat_table_header`) + `Origin Anchor` (`page_no` + `source_url` + `sha256`)

```
import hashlib
from datetime import datetime, timezone
from docling.chunking import HybridChunker

chunker = HybridChunker(tokenizer="sentence-transformers/all-MiniLM-L6-v2", max_tokens=512, repeat_table_header=True)
records = []
for idx, chunk in enumerate(chunker.chunk(doc)):
    contextual_text = chunker.contextualize(chunk) # Automatically prepends parent section headings!
    page_no = chunk.meta.doc_items[0].prov[0].page_no if chunk.meta.doc_items else None
    records.append({
        "id": f"{doc_id}_{idx}", "text": contextual_text,
        "headings": chunk.meta.headings, "page_no": page_no,
        "source_url": source_uri, "sha256": hashlib.sha256(contextual_text.encode()).hexdigest(),
        "ingested_at": datetime.now(timezone.utc).isoformat(),
    })
```

1 Why `HybridChunker` Beats Fixed Splits

Operating directly on `DoclingDocument`, `HybridChunker` merges tiny adjacent bullets, splits oversized sections along sentence boundaries using your embedding model tokenizer, and repeats table headers if large tables span chunks.

2 Audit-Grade Citation Provenance

Calling `chunker.contextualize(chunk)` embeds parent headings right above the chunk body for vector search, while `page_no` and bounding coordinates let agents render exact page links in regulated workflows.

Recommended Chunking Profiles by Document Workload

Workload Type	Token Target	Header & Table Strategy	Mandatory Metadata Fields
Developer Docs & APIs	`384 – 512` tok	Keep fenced code blocks intact; prepend H1/H2/H3 breadcrumb.	`source_url`, `headings`, `doc_version`, `sha256`
Financial Filings & PDFs	`512 – 768` tok	`TableFormer ACCURATE` + `repeat_table_header=True`.	`file_name`, `page_no`, `bbox`, `fiscal_period`
Web Knowledge Portals	`400 – 600` tok	Markdown header split on `fit_markdown` after boilerplate prune.	`canonical_url`, `last_modified`, `sha256`
Legal & Policy Contracts	`256 – 400` tok	Clause-level split preserving numbered section hierarchy.	`clause_id`, `page_no`, `effective_date`

Five Copy-Paste Production Ingestion Recipes

Battle-tested patterns combining Docling, Crawl4AI, and Firecrawl across common engineering tasks—from local annual report extraction to live webhook-driven monitoring.

Recipe & Goal	Recommended Stack	Core Implementation Pattern	Cost & Reliability Gate
1. Financial Table Harvester Extract multi-page PDF tables	Docling + `TableFormer` + Pydantic	`opts.table_structure_options.mode = TableFormerMode.ACCURATE` `chunks = HybridChunker(max_tokens=512).chunk(doc)`	Zero cloud token fees; validate extracted row sums deterministically.
2. Zero-LLM Product Catalog Scrape 1,000+ listing URLs	Crawl4AI + `JsonCssExtraction`	`strategy = JsonCssExtractionStrategy(catalog_schema)` `await crawler.arun_many(urls, config=run_cfg)`	`MemoryAdaptiveDispatcher` caps browser RAM below 75%.
3. Authenticated SPA Docs Crawl portal behind login	Crawl4AI + `session_id` + Hooks	Set auth cookies in `on_page_context_created`; click tabs with `js_code` & `js_only=True`.	Explicitly call `kill_session(session_id)` in `finally` block.
4. Live Pricing Diff Monitor Alert only when tiers change	Firecrawl v2 + `changeTracking`	`formats=["markdown", {"type": "changeTracking", "modes": ["git-diff"]}]`; check `changeStatus`.	Pass `max_age=0` to bypass 2-day cache on live monitors.
5. Coding Agent Web Tool Live API docs inside IDE	`firecrawl-mcp` + Claude / Cursor	Agent calls `firecrawl_map(url)` then `firecrawl_scrape(target)` for pruned docs.	Whitelist approved doc domains; wrap output in nonce tags.

Recipe 2 Implementation: Resilient Concurrent Batch Crawl with Rate Limiter

```
import asyncio
from crawl4ai import AsyncWebCrawler, BrowserConfig, CrawlerRunConfig, CacheMode, RateLimiter
from crawl4ai.async_dispatcher import MemoryAdaptiveDispatcher
from crawl4ai.content_filter_strategy import PruningContentFilter
from crawl4ai.markdown_generation_strategy import DefaultMarkdownGenerator

async def batch_ingest_docs(urls: list[str]) -> dict[str, str]:
    dispatcher = MemoryAdaptiveDispatcher(memory_threshold_percent=75.0, max_session_permit=6,
        rate_limiter=RateLimiter(base_delay=(1.0, 2.0), max_delay=30.0, max_retries=3))
    run_cfg = CrawlerRunConfig(cache_mode=CacheMode.ENABLED,
        markdown_generator=DefaultMarkdownGenerator(content_filter=PruningContentFilter(threshold=0.45)))
    async with AsyncWebCrawler(config=BrowserConfig(headless=True, verbose=False)) as crawler:
        results = await crawler.arun_many(urls=urls, config=run_cfg, dispatcher=dispatcher)
        return {r.url: r.markdown.fit_markdown for r in results if r.success and r.markdown.fit_markdown}
```

Pro Tip: Pair `firecrawl.map()` with `Crawl4AI` Batch Workers
Use Firecrawl `/map` once to obtain a clean, sitemap-verified list of documentation URLs in under two seconds, then feed that URL batch into local Crawl4AI `arun_many()` workers for free, high-speed Markdown pruning and custom CSS extraction.

Crawl Ethics, Prompt-Injection Quarantine & Deduplication

External websites and third-party PDFs are untrusted inputs. Production pipelines respect domain crawl rules, sanitize invisible adversarial instructions deterministically, and hash normalized text before embedding.

1 Politeness & `robots.txt`

Check `robots.txt` rules before crawling, honor `Crawl-delay` and HTTP `429 Retry-After` headers, send an identifiable `User-Agent` contact string, and cache repeat fetches cleanly.

2 Prompt-Injection Defense

Attackers embed hijacks inside `display:none` elements, HTML comments, `opacity:0` spans, or invisible Unicode tags (`U+200B`, `U+E0000`). Scrub them prior to chunking.

3 `SHA-256` & Near-Dedup

Canonicalize URLs (strip `utm_*` params), normalize whitespace, and skip identical `SHA-256` hashes. Add MinHash or SimHash LSH to catch near-duplicate mirror pages.

Production Sanitizer, SHA-256 Deduplicator & Nonce Quarantine Wrapper

```
import hashlib, re, secrets, unicodedata
from bs4 import BeautifulSoup, Comment

INVISIBLE_RE = re.compile(r"[\u200b-\u200f\ufeff\u202a-\u202e\U000e0000-\U000e007f]")
HIDDEN_STYLE_RE = re.compile(r"(display\s*:\s*none|visibility\s*:\s*hidden|opacity\s*:\s*0|font-size\s*:\s*0)", re.I)

def sanitize_and_digest(raw_html: str) -> tuple[str, str]:
    soup = BeautifulSoup(raw_html, "html.parser")
    for tag in soup(["script", "style", "noscript", "template"]): tag.decompose()
    for c in soup.find_all(string=lambda t: isinstance(t, Comment)): c.extract()
    for hidden in soup.find_all(style=HIDDEN_STYLE_RE): hidden.decompose()
    clean = INVISIBLE_RE.sub("", unicodedata.normalize("NFKC", soup.get_text(" ", strip=True)))
    digest = hashlib.sha256(re.sub(r"\s+", " ", clean).strip().encode("utf-8")).hexdigest()
    return clean, digest

def frame_for_rag(clean_text: str, source_url: str) -> str:
    nonce = secrets.token_hex(8)
    return f'[UNTRUSTED_WEB_SOURCE url="{source_url}" nonce="{nonce}"]\n{clean_text}\n[/UNTRUSTED_WEB_SOURCE nonce="
{nonce}"]'
```

U URL Canonicalization Rules

Normalize scheme and lowercase host, strip fragment anchors (`#section`), drop tracking parameters (`utm_*`, `fbclid`, `gclid`), and sort remaining query keys so identical pages never enter the crawl queue twice.

H Exact Hash vs. Fuzzy Similarity

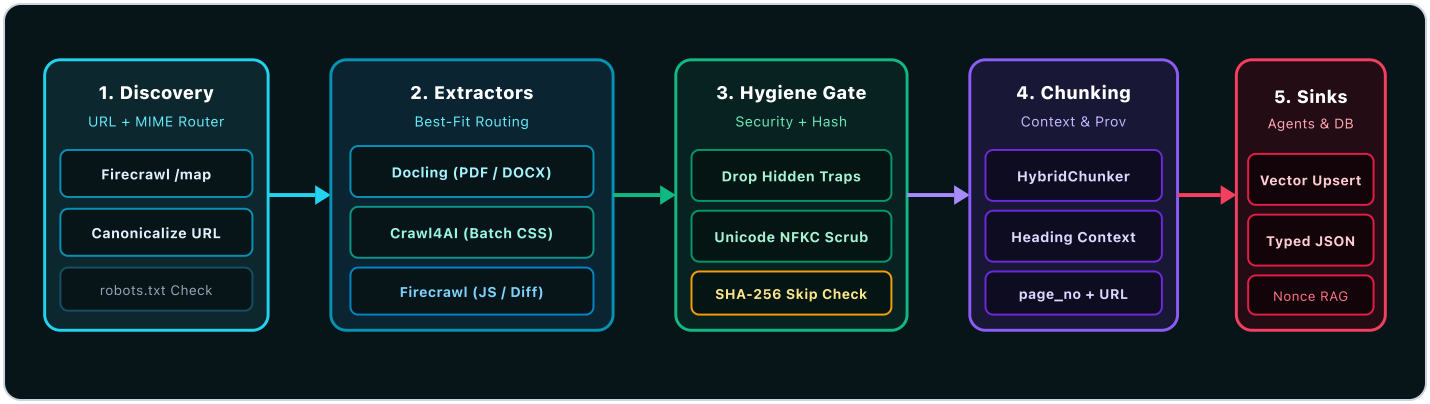
Use exact `SHA-256` digests to skip byte-identical re-crawls at zero cost. Layer 128-permutation MinHash or SimHash LSH above that to collapse regional mirror pages that differ only by currency or timestamp banners.

Defense in Depth: Never Rely on System Prompts Alone

Telling an agent to ignore instructions found in scraped text fails whenever adversarial directives share the same token stream. Combine deterministic pre-ingestion DOM and Unicode sanitization with cryptographic nonce framing, output-schema checks, and strict sandbox egress controls so retrieved text remains inert data.

End-to-End Synergy Pipeline: Discovery to Verified RAG

Rather than forcing one engine onto every workload, production teams route web links, interactive portals, and binary attachments to specialized extractors before merging everything into one clean index.



1 Hybrid Web + PDF Documentation Portals

Corporate knowledge portals constantly mix HTML pages with linked `.pdf` whitepapers and specification sheets. Configure Firecrawl `/map` or Crawl4AI to collect all links, route HTML pages through `PruningContentFilter`, and pass discovered `.pdf` and `.docx` URLs directly to `Docling` `DocumentConverter` so tables never flatten.

2 Webhook-Driven Incremental Syncs

Rather than re-scraping and re-embedding your entire corpus nightly, attach Firecrawl `changeTracking` webhooks to trigger re-chunking only when `changeStatus` equals `changed` or `new`. Pair that gate with chunk-level `SHA-256` hashing to skip unchanged paragraphs and cut embedding costs by over 80%.

Ecosystem Handoffs: MCP, Vector Stores & Eval Harnesses

M Live Agent Tooling

Expose `firecrawl-mcp` or Docling tools via MCP so coding agents pull fresh API docs and attachments on demand.

V Hybrid Vector Search

Index dense embeddings alongside sparse BM25 vectors and filter on `headings` and `source_url` metadata.

E Golden Regression Sets

Keep 20 tricky PDFs and SPAs in CI to benchmark TableFormer accuracy and Markdown pruning on every upgrade.

Four Pipeline SLIs Every Ingestion Team Should Track

Measure four numbers on every run: (1) **Extraction success rate** across MIME types, (2) **Markdown compression ratio** comparing raw HTML to `fit_markdown`, (3) **Quarantine strip count** logging blocked hidden prompts, and (4) **SHA-256 skip rate** confirming incremental sync efficiency.

Troubleshooting Clinic & Pre-Flight Checklist

Rapid fixes for the seven most frequent document parsing and web scraping failure modes, paired with a mandatory six-point pre-flight check before launching production ingestion jobs.

Symptom & Error	Root Cause	10-Second Production Fix
`fit_markdown` is empty string	`DefaultMarkdownGenerator` initialized without a content filter.	Attach `PruningContentFilter(...)` or `BM25ContentFilter(...)` to `markdown_generator`.
TableFormer scrambles merged columns	Default `FAST` table mode missed multi-row or hierarchical headers.	Set `opts.table_structure_options.mode = TableFormerMode.ACCURATE` or try `VlmPipeline`.
Scanned PDF outputs blank text	File holds raster page images with no text layer and OCR is off.	Enable `do_ocr=True` with `RapidOcrOptions` or `MacOcrOptions` (or `force_full_page_ocr=True`).
React / SPA scrape misses body	Extractor read the DOM before client hydration or API fetch completed.	Add `wait_for="css:.article-body"` in Crawl4AI or `actions=[{"type": "wait", ...}]` in Firecrawl.
`arun_many` triggers OOM crash	Too many headless Chromium contexts opened concurrently on host RAM.	Use `MemoryAdaptiveDispatcher(memory_threshold_percent=75.0)` and cap `max_session_permit`.
Firecrawl returns stale page content	Firecrawl v2 defaults to a 2-day semantic cache window (`maxAge`).	Pass `max_age=0` on live price, inventory, or changelog checks that require fresh fetches.
Duplicate chunks pollute RAG	Tracking query strings (`?utm_source=`) or unchanged re-crawls re-embedded.	Strip marketing query params during URL canonicalization and skip identical `SHA-256` digests.

6-Point Pre-Flight Ingestion Readiness Checklist

☐ **1. Robots & Rate Limits Active:** Verify `robots.txt` compliance, exponential backoff via `RateLimiter` on HTTP 429/503, and an honest `User-Agent` string.

☐ **2. Boilerplate Pruning Verified:** Sample raw HTML versus `fit_markdown` output to confirm navbars, cookie modals, and footers are stripped cleanly.

☐ **3. Table & OCR Fidelity Tested:** Run complex golden-set PDFs through `TableFormerMode.ACCURATE` and check numeric column alignment.

☐ **4. Injection Sanitizer Armed:** Test that `display:none` blocks, HTML comments, and `U+200B` tokens are removed and RAG context uses nonces.

☐ **5. SHA-256 Dedup & Canonical URLs:** Confirm normalized digests skip unchanged re-ingests and tracking query parameters are stripped.

☐ **6. Provenance Metadata Populated:** Check that every chunk in your vector database carries `source_url`, `headings`, `page_no`, and `ingested_at`.