

# EXAMTRACKER INDIA

## PDF Parsing Pipeline & Data Schema

Architecture Reference Document • February 2026 • Version 1.0

## Overview: What This Pipeline Does

When the scraper detects a new notification URL on any of the 142 monitored sites, it downloads the PDF. That PDF is a raw document in one of many unpredictable formats. The pipeline's job is to convert that raw PDF into a clean, structured row in the **exams** PostgreSQL table — with every eligibility field correctly populated so the matching engine can run against it.

The core challenge: Indian government recruitment PDFs have **zero standardisation**. A PDF from SSC looks completely different from UPSC, which looks different from a State PSC, which differs from RRB. The pipeline must handle all of them correctly and **never publish data it is not confident about**.

**KEY RULE** An exam record is never shown to users until it clears human verification. The pipeline creates a DRAFT — a human editor approves it. Wrong data is worse than no data.

## 1. PDF Type Taxonomy

Before any processing can start, the pipeline must identify what kind of PDF it is dealing with. Every subsequent step depends on this classification.

Type	Name	Who Uses This	Processing Approach	Difficulty
Type A	Digital Text PDF	SSC, UPSC, IBPS, SBI, most banking	Direct text extraction — fastest, most accurate	Easy
Type B	Scanned Image PDF	Many State PSCs, older RRB notifications, Gazette PDFs	OCR required — slower, needs confidence scoring	Medium
Type C	Mixed (digital + scanned pages)	Common in long notifications where main body is typed but annexures are scanned	Page-level detection: some pages get extraction, some get OCR	Medium
Type D	Hindi/Regional language only	State PSCs in Hindi belt, UP, Bihar, MP, Rajasthan	OCR with Devanagari model + translation pipeline	Hard
Type E	Bilingual (English + Hindi)	GOI Press Gazette format, CRPF, BSF, many central bodies	Extract English blocks only, ignore Hindi blocks (parallel text)	Medium

Type	Name	Who Uses This	Processing Approach	Difficulty
Type F	Table-heavy / Excel-converted	RRB vacancy annexures, SSC category-wise vacancy tables	Table extraction engine (Camelot/pdfplumber) — treats tables differently from prose	Medium

## The 7-Stage Pipeline

Every PDF that enters the system passes through all 7 stages in sequence. A stage can either **pass the PDF forward**, **flag it for human review**, or **reject it entirely**. Nothing skips stages.

### Pipeline Flow at a Glance

#	Input	What Happens	Output	Primary Tool
1	URL from scraper	<b>ACQUISITION</b> — Download PDF, deduplicate, store raw bytes	Raw PDF in object storage	<i>Node.js + Supabase Storage</i>
2	Raw PDF	<b>CLASSIFICATION</b> — Detect PDF type (A–F), language, page count	Type tag + metadata	<i>pdfjs-dist + pytesseract probe</i>
3	Typed PDF	<b>EXTRACTION</b> — Pull raw text using method matching the type	Raw text blocks per page	<i>pdfplumber / Camelot / Tesseract</i>
4	Raw text blocks	<b>PARSING</b> — Identify sections, find key fields in text	Labelled field candidates	<i>Regex rules + Claude Haiku</i>
5	Field candidates	<b>STRUCTURING</b> — Map fields to DB schema, resolve ambiguity	Candidate JSON object	<i>Rules engine + Haiku validation</i>
6	Candidate JSON	<b>CONFIDENCE SCORING</b> — Score each field, flag low-confidence	Scored JSON + review flags	<i>Scoring function (per-field rules)</i>
7	Scored JSON	<b>HUMAN REVIEW</b> — Editor sees flagged fields, approves/corrects	Verified exam record → DB	<i>Admin dashboard</i>

### STAGE 1

#### Acquisition

*Download, deduplicate, store the raw PDF*

The scraper has already discovered a URL. Acquisition handles getting the actual PDF and making sure we don't process the same document twice.

#### What happens:

- Download PDF using Node.js fetch with a 30s timeout. If the server returns HTML instead of a PDF (common when sites require login), flag as SCRAPE\_BLOCKED.
- Compute SHA-256 hash of the raw PDF bytes immediately on download.
- Check the hash against a pdf\_hashes table in PostgreSQL. If already seen, discard — this is a duplicate (same PDF re-linked from a new URL, very common in government sites).
- Store the raw PDF in Supabase Storage at path: /raw-pdfs/{source\_id}/{YYYY-MM}/{hash}.pdf

- Create a pdf\_ingestion\_log record: source\_id, url, hash, download\_timestamp, file\_size\_bytes, status = DOWNLOADED.

**WHY HASH?** The same SSC CGL notification PDF is often linked from 9 different SSC regional office websites. Without deduplication, you would parse and create 9 duplicate exam records. Hash-based dedup at acquisition prevents all downstream wasted work.

## STAGE 2

### Classification

*Detect PDF type, language, structure*

Before extracting anything, the pipeline must know what it is dealing with. Classification answers three questions: Is the PDF machine-readable or scanned? What language is it in? How is the content structured?

#### Detection logic:

- Open PDF with pdfjs-dist (JavaScript) or pdfplumber (Python). Attempt to extract text from page 1.
- If extracted text is > 50 characters and looks like English/Hindi words → Type A (digital text). Mark as TEXT\_EXTRACTABLE.
- If extracted text is < 50 characters or is garbage unicode → probe for embedded images. If images found → Type B (scanned). Mark as NEEDS\_OCR.
- If some pages are extractable and some are not → Type C (mixed). Process page by page.
- Run a quick language probe on the first 500 characters of extracted text: if Devanagari character ratio > 60% → Type D (Hindi-only). If mixed → Type E (bilingual).
- Run Camelot's table detection on first 3 pages. If > 2 tables detected → flag as TABLE\_HEAVY (affects extraction strategy in Stage 3).

#### Output stored in pdf\_ingestion\_log:

- pdf\_type: ENUM (TEXT / SCANNED / MIXED / HINDI\_ONLY / BILINGUAL / TABLE\_HEAVY)
- language\_detected: ENUM (ENGLISH / HINDI / BILINGUAL / REGIONAL)
- page\_count: integer
- has\_tables: boolean
- ocr\_required: boolean

## STAGE 3

### Extraction

*Pull raw text using the method matching the PDF type*

Extraction converts the PDF into raw text blocks. Three different methods are used depending on classification. The goal is maximum fidelity — get everything out, worry about making sense of it in Stage 4.

PDF Type	Method	Library	Notes
Type A (Digital)	Direct text extraction	<a href="#">pdfplumber (Python)</a>	Preserves reading order. Extract word-level bounding boxes for later table reconstruction. Fast — < 1s per PDF.
Type B (Scanned)	OCR via Tesseract	<a href="#">pytesseract + Pillow</a>	Convert pages to 300 DPI images first. Use eng+hin language pack. Confidence score per word — words below 60% confidence are marked uncertain.
Type C (Mixed)	Page-level routing	<a href="#">pdfplumber + pytesseract</a>	Each page goes to the right method independently. Results merged in reading order.

PDF Type	Method	Library	Notes
Type D (Hindi only)	OCR with Devanagari	<i>pytesseract (hin mode)</i>	Extract Hindi text, then translate key fields via Google Translate API. Only translate dates, numbers, qualification terms — not full text.
Type E (Bilingual)	Column detection	<i>pdfplumber</i>	Detect two-column layout (very common in GOI Gazette format). Extract only the English column based on x-coordinates. Discard Hindi column.
Type F (Table-heavy)	Table extraction	<i>Camelot (lattice mode)</i>	Extract all tables as pandas DataFrames. Preserve cell structure including merged cells. Treat table cells as labelled values, not free text.

**IMPORTANT** Extraction output is stored as raw text in `pdf_ingestion_log` (`extracted_text` column, TEXT type). Never throw away the raw extraction. If the parser fails in Stage 4, a human can re-run parsing on the stored extracted text without re-downloading the PDF.

## STAGE 4

### Parsing

*Identify sections and extract key fields from raw text*

Parsing is the most complex stage. It takes the raw extracted text and tries to find specific values: exam name, dates, vacancies, eligibility rules. It uses a two-layer approach: **regex rules first**, Claude Haiku only for what regex cannot reliably handle.

### Layer 1: Regex + Pattern Rules (handles ~70% of fields)

Dates, fees, vacancy numbers, age limits, and application windows all follow predictable patterns across all PDFs. These are handled by a library of pre-written regex patterns:

Field	Regex Pattern (simplified)	Notes
Application end date	"Last Date.*?:(\s*)(\d{2}[-/.]\d{2}[-/.]\d{4})"	Covers most GOI date formats
Age limit (general)	Catches "18 to 27 years" pattern	
OBC age relaxation	"OBC.*?(\d+)\s*years?\s*relaxation"	Catches "OBC: 3 years relaxation"
Total vacancies	Works for most formats	
Application fee (General)	"[Gg]eneral.*?Rs.\.?s*(\d+)"	Catches Rs.100, Rs 500 etc.
Qualification (graduation)	"[Gg]raduat [Bb].? [Ee][Bb].? [Tt]ech [Bb].?[Ss]c"	Qualification keyword matching
Exam date	"[Ee]exam.*?/[Dd]ate.*?:?\s*(\w+\d{4}) \d{2}/\d{2}/\d{4})"	Handles both formats
Notification number	"[Aa]dvt.?\s*[Nn]o.?.*?([A-Z0-9/-]+)\d{4})"	For deduplication

### Layer 2: Claude Haiku LLM (handles the remaining ~30%)

Some fields cannot be reliably extracted with regex because they are expressed in varied natural language. For these fields only, the extracted text (or specific sections of it) is sent to Claude Haiku with a structured prompt asking for a JSON response.

**FIELDS SENT TO HAIKU**

Post names and their individual eligibility (when one PDF covers 20+ different posts with different rules per post) | Ex-Serviceman age relaxation rules buried in paragraphs | State-specific domicile requirements | Physical standards requirements (height/vision) | Marital status requirements | Complex qualification combinations (e.g. "B.Tech in Civil/Mechanical/Electrical/Electronics OR Diploma with 3 years experience")

**Haiku Prompt Design (critical):**

- Always send a structured schema in the system prompt showing exactly what JSON shape to return.
- Send only the relevant section of the PDF text, not the entire document. Use section detection (look for section headings like "Eligibility", "Age Limit", "Educational Qualification") to crop the text before sending.
- Always include: "If you cannot determine a field with certainty, return null for that field. Never guess."
- Use temperature = 0 for deterministic extraction. This is not creative work.
- Cap text sent to Haiku at 2000 tokens per call. For long PDFs, split into section-specific calls.

**STAGE 5****Structuring**

*Map parsed fields to the database schema*

Structuring takes the extracted field values and maps them into the exact shape of the **exams** table. This is where ambiguities are resolved, defaults are applied, and multi-post PDFs are split into individual exam records.

**Key structuring decisions:**

- Multi-post PDFs: If parsing found multiple post names (e.g. SSC CGL has 20+ posts), create one candidate exam record per post. Each post gets its own vacancies\_by\_category, age limits, and qualification requirement.
- Vacancy breakdown: If only total vacancies were found (not category-wise), set total\_vacancies and leave vacancies\_by\_category as null. Do not estimate category splits.
- Age limit defaults: If only "age as per GOI rules" is stated without specific numbers, look up the exam body's standard rules from the age\_relaxation\_rules reference table (maintained separately). Apply them but flag as INFERRED not EXTRACTED.
- Qualification normalisation: Map free text like "graduation in any discipline" to qualification\_level\_enum = GRADUATION. Map "B.Tech/B.E." to GRADUATION\_ENGINEERING.
- Date parsing: All dates normalised to ISO 8601 (YYYY-MM-DD). If only month+year found (e.g. "March 2026"), set to the first of that month and flag as APPROXIMATE.
- Fee mapping: Map "Nil for SC/ST" to application\_fee\_sc\_st = 0. Map "exempted" to 0. Map "same as General" to the General fee value.

**STAGE 6****Confidence Scoring**

*Score each field, decide what needs human review*

Every field in the candidate exam record gets a confidence score from 0.0 to 1.0. The overall record score is the average of all required field scores. If the overall score is below the threshold, the entire record is flagged for mandatory human review.

Field	Priority	Auto-approve threshold	Best case scenario	Score	Partial extraction	Score	Not found
application_end	CRITICAL	≥ 0.90 to auto-approve	Extracted by regex from clear date pattern	1.0	Found by Haiku from ambiguous text	0.7	Not found at all
exam_name	HIGH	≥ 0.85	Exact match to known exam names in DB	1.0	Haiku extraction	0.8	Not found

Field	Priority	Auto-approve threshold	Best case scenario	Score	Partial extraction	Score	Not found
total_vacancies	HIGH	$\geq 0.80$	Regex match with clear number	0.95	Haiku extraction	0.75	Not found
vacancies_by_category	MEDIUM	$\geq 0.70$	Complete table extracted	0.95	Partial table	0.6	Not in PDF
age_limits	CRITICAL	$\geq 0.90$	Explicit numbers extracted	1.0	Inferred from GOI rules	0.7	Not found
required_qualification	HIGH	$\geq 0.80$	Clear qualification stated	0.9	Ambiguous statement	0.6	Not found
application_fee	LOW	$\geq 0.60$	Fee table extracted	0.9	Partial fee info	0.6	Not found

**Auto-approve rule:** If all CRITICAL fields score  $\geq$  their threshold AND overall average  $\geq 0.85 \rightarrow$  record goes to PENDING\_HUMAN\_SPOT\_CHECK (editor sees it but is not blocked). If any CRITICAL field < threshold  $\rightarrow$  record is BLOCKED\_FOR REVIEW (must be approved before going live).

## STAGE 7

### Human Review

*Editor verifies, corrects, and approves the record*

No exam record goes live without at least one human seeing it. The admin dashboard shows the editor exactly what was extracted, what confidence score each field received, and what the raw PDF says. The editor can approve as-is, edit individual fields, or reject the record entirely.

#### What the admin dashboard shows per field:

- Extracted value (what the pipeline found)
- Confidence score (colour-coded: green  $> 0.9$ , orange 0.7–0.9, red  $< 0.7$ )
- Source snippet: the exact text from the PDF that this value was extracted from (so editor can verify without opening the PDF)
- Extraction method: REGEX / HAIKU / INFERRRED / NOT\_FOUND
- Quick-edit inline: editor can correct any field value directly in the dashboard

#### Editor time estimate:

- High-confidence record (auto-approved): spot-check takes 30 seconds — scan the fields, click Approve.
- Medium-confidence record: 2–3 minutes — review flagged fields, compare with PDF, fix and approve.
- Low-confidence / complex record: 5–8 minutes — open original PDF, manually correct multiple fields.

#### SPEED TARGET

At MVP scale (50 new notifications/day), an editor doing 2 hours of review work can process all records. As automation improves and confidence scores rise (template library grows), auto-approve rate increases and editor time per record drops.

## The Extracted Data Schema

This is the complete set of fields the pipeline tries to fill from every PDF. These map directly to columns in the **exams** PostgreSQL table. Fields marked YES in Required are checked at human review — a record cannot go live with these missing.

### Group 1: Core Identity

Field Name	Data Type	Source in PDF	Required?	Notes
<code>exam_name</code>	<code>VARCHAR(300)</code>	Title of notification, first page heading	<b>YES</b>	e.g. "Combined Graduate Level Examination 2025"
<code>short_name</code>	<code>VARCHAR(100)</code>	Abbreviation in notification text	NO	e.g. "SSC CGL 2025" — inferred if not stated
<code>slug</code>	<code>VARCHAR(200)</code>	Generated by system	SYSTEM	e.g. "ssc-cgl-2025" — auto-generated from name + year
<code>conducting_body</code>	<code>VARCHAR(200)</code>	Issuing authority name in header/footer	<b>YES</b>	e.g. "Staff Selection Commission"
<code>notification_number</code>	<code>VARCHAR(100)</code>	Advt. No. / Notification No. field	NO	Used for deduplication of corrigenda
<code>category</code>	<code>ENUM</code>	Matched from conducting_body to category list	<b>YES</b>	SSC / RAILWAY / BANKING / etc.
<code>level</code>	<code>ENUM</code>	Derived from conducting_body	<b>YES</b>	CENTRAL / STATE / PSU / DEFENCE / BANKING
<code>state_code</code>	<code>ENUM / NULL</code>	State mentioned in notification header	CONDITIONAL	NULL for national exams. Required for state exams.

### Group 2: Key Dates

Field Name	Data Type	Source in PDF	Required?	Notes
<code>notification_date</code>	<code>DATE</code>	Date on notification PDF / publication date	NO	Date the notification was officially released
<code>application_start</code>	<code>DATE</code>	"Application begins / Registration opens" section	NO	Some notifications skip this — that's acceptable
<code>application_end</code>	<code>DATE</code>	"Last date for application / Closing date" section	<b>YES</b>	CRITICAL — primary alert trigger. Record blocked without this.
<code>fee_payment_end</code>	<code>DATE</code>	"Last date for fee payment" — sometimes differs from app end	NO	Often same as application_end but not always
<code>exam_date</code>	<code>DATE / NULL</code>	"Date of examination" section	NO	Often announced separately — null is fine at notification stage

Field Name	Data Type	Source in PDF	Required?	Notes
admit_card_date	DATE / NULL	"Admit card download" section	NO	Trigger for admit card reminder notification
result_date	DATE / NULL	"Result declaration" section	NO	Trigger for result notification to users
age_cutoff_date	DATE	"Age as on [date]" — often specified explicitly	YES	Critical for accurate age eligibility calculation

### Group 3: Vacancy Data (Core Differentiator for Vacancy-Aware Mode)

Field Name	Data Type	Source in PDF	Required?	Notes
total_vacancies	INTEGER	Total vacancy number — usually prominently stated	YES	Extract the total, not sum of category rows (avoid double-count)
vacancies_by_category	JSONB	Vacancy table with UR/OBC/SC/ST/EWS/PwBD/ESM columns	NO	The key field for VACANCY_AWARE mode. Null is acceptable if not in PDF.
post_name	VARCHAR(200)	Name of specific post (when PDF covers multiple posts)	YES	One exam record per post for multi-post PDFs
post_code	VARCHAR(50)	Post code if listed in vacancy table	NO	Useful for linking corrigenda to original notification

#### VACANCY TABLE CHALLENGE

Government PDFs present vacancy data in 4 different ways: (1) One table with category columns — ideal, extract directly. (2) Total only, no breakdown — store total, set vacancies\_by\_category = null. (3) Breakdown in a separate annexure PDF — link annexure via notification\_number, process as related record. (4) "As per roster" with no numbers — set total\_vacancies = null, flag for human research.

### Group 4: Age Eligibility (Stored Explicitly, Not Computed)

Field Name	Data Type	Source in PDF	Required?	Notes
min_age	SMALLINT	"Minimum age" / "Not below X years" text	YES	Almost always 18. Flag if different.
max_age_general	SMALLINT	Age limit for UR/General category	YES	Base age limit — typically 25–35 depending on exam
max_age_abc	SMALLINT	OBC limit = General + 3 (or as stated)	YES	Extract explicitly from PDF. Do not auto-compute.
max_age_sc_st	SMALLINT	SC/ST limit = General + 5 (or as stated)	YES	Extract explicitly — some exams differ from standard
max_age_ews	SMALLINT	"EWS age limit" — usually same as General	PARTIAL	Null if not mentioned — will be treated same as General

Field Name	Data Type	Source in PDF	Required?	Notes
<code>max_age_pwd_general</code>	<code>SMALLINT</code>	"PwBD/PWD age relaxation" section	NO	<i>General + 10 is standard but extract explicitly</i>
<code>max_age_ex_serviceman</code>	<code>SMALLINT</code>	"Ex-Serviceman relaxation" section	NO	<i>Complex rules — flag for human if found but unclear</i>
<code>age_basis</code>	<code>VARCHAR(100)</code>	"Age as on [specific date]" statement	<b>YES</b>	<i>The specific cutoff date — critical for accurate calculation</i>

#### Group 5: Educational Qualification

Field Name	Data Type	Source in PDF	Required?	Notes
<code>required_qualification</code>	<code>ENUM</code>	Qualification section / eligibility section	<b>YES</b>	<i>Map to: CLASS_10 / CLASS_12 / ITI / DIPLOMA / GRADUATION / GRADUATION_ENGINEERING / POST_GRADUATION / etc.</i>
<code>required_streams</code>	<code>TEXT[]</code>	Stream/discipline requirements	NO	<i>e.g. ["Civil", "Mechanical", "Electrical"] — null means any stream</i>
<code>min_marks_percentage</code>	<code>NUMERIC(5,2)</code>	"Minimum X% marks" in qualification text	NO	<i>Only required for RBI Grade B, SBI PO, NABARD etc. Most exams: null</i>
<code>allows_final_year</code>	<code>BOOLEAN</code>	"Final year students may apply" text	NO	<i>Default false if not mentioned</i>
<code>additional_qualification_required</code>	<code>TEXT</code>	Additional certificates needed	NO	<i>e.g. "B.Ed required for teaching posts", "NCC C certificate preferred"</i>

#### Group 6: Other Eligibility Criteria

Field Name	Data Type	Source in PDF	Required?	Notes
<code>nationality_requirement</code>	<code>ENUM</code>	"Citizenship / Nationality" section	<b>YES</b>	<i>Default: INDIAN. Extract if Nepal/Bhutan/OCI allowed.</i>
<code>gender_restriction</code>	<code>ENUM / NULL</code>	"Only male/female candidates" explicit statement	NO	<i>NULL = no restriction. Only extract if explicitly stated.</i>
<code>physical_requirements</code>	<code>JSONB / NULL</code>	Physical standards section	NO	<i>{"height_general_male": 170, "height_sc_male": 165, "vision": "6/6"}</i>
<code>marital_status_requirement</code>	<code>ENUM / NULL</code>	Marital status section (rare)	NO	<i>Only for NDA (unmarried) and some state exams</i>

Field Name	Data Type	Source in PDF	Required?	Notes
<code>domicile_required</code>	<code>BOOLEAN</code>	"Domicile of [state]" clause	CONDITIONAL	<i>True for all state-level exams</i>

### Group 7: Application Fees

Field Name	Data Type	Source in PDF	Required?	Notes
<code>application_fee_general</code>	<code>NUMERIC(8,2)</code>	Fee table — General/UR row	NO	e.g. 100.00
<code>application_fee_sc_st</code>	<code>NUMERIC(8,2)</code>	Fee table — SC/ST row	NO	Often 0 — exempted
<code>application_fee_pwd</code>	<code>NUMERIC(8,2)</code>	Fee table — PwBD/PWD row	NO	Often 0 — exempted
<code>application_fee_women</code>	<code>NUMERIC(8,2)</code>	Fee table — Female row (some exams)	NO	<i>Rare but exists in some state exams</i>
<code>application_fee_obc</code>	<code>NUMERIC(8,2)</code>	Fee table — OBC row if different	NO	<i>Usually same as General — extract if explicitly different</i>
<code>fee_payment_modes</code>	<code>TEXT[]</code>	"Mode of payment" section	NO	e.g. ["Online", "UPI", "Debit Card"] — for document checklist

### Group 8: Trust & Verification Metadata

Field Name	Data Type	Source in PDF	Required?	Notes
<code>official_notification_url</code>	<code>TEXT</code>	The URL the scraper found this PDF at	<b>YES</b>	<i>Already known from Stage 1 — not extracted from PDF</i>
<code>notification_verified</code>	<code>BOOLEAN</code>	Set to TRUE by human editor on approval	<b>YES</b>	<i>Default FALSE until editor approves</i>
<code>data_source</code>	<code>ENUM</code>	Set by pipeline automatically	SYSTEM	<i>MANUAL / AUTO_TEXT / AUTO_OCR / AUTO_HAIKU</i>
<code>extraction_confidence</code>	<code>NUMERIC(3,2)</code>	Overall confidence score from Stage 6	SYSTEM	<i>0.00 to 1.00</i>
<code>needs_human_review</code>	<code>BOOLEAN</code>	Set by Stage 6 scoring engine	SYSTEM	<i>TRUE if any CRITICAL field &lt; threshold</i>
<code>review_flags</code>	<code>JSONB</code>	Array of per-field flags from Stage 6	SYSTEM	<code>{"application_end": "LOW_CONFIDENCE", "age_limits": "INFERRRED"}</code>
<code>last_verified_at</code>	<code>TIMESTAMPTZ</code>	Set by human editor on approval	SYSTEM	<i>Used to detect stale records needing re-verification</i>

# The Template Library: How Accuracy Improves Over Time

The single most important investment in the parsing layer is building and growing a Template Library — a collection of source-specific parsing rules that override generic parsing for known document formats.

Every government body reuses the same document format across years. SSC CGL 2025 looks almost identical to SSC CGL 2024. Once you parse one correctly, you can parse all future ones with near-100% accuracy.

Template ID	Covers	Confidence after 3 examples	Key parsing rules
SSC_CENTRAL	SSC CGL, CHSL, MTS, CPO, JE, Steno	~97%	Look for "IMPORTANT DATES" table in first 2 pages. Category vacancy table always has UR/OBC/SC/ST/EWS/ESM/PwBD as column headers. Age in "GENERAL INSTRUCTIONS" section.
UPSC_CIVIL	IAS, IPS, IFS Civil Services	~95%	Single-post, no vacancy breakdown table. Age in Part III. Application window always 21 days from notification date.
RRB_NTPC	RRB NTPC, Group D, ALP	~96%	Bilingual PDF. Vacancy table in Annexure at end. Age cutoff always 01-Jan of exam year. 21 RRB sites but identical format.
IBPS_BANK	IBPS PO, Clerk, SO, RRB Banks	~98%	Very consistent format year over year. Fee table always on page 2. Dates in a boxed Important Dates section near top.
STATE_PSC_UP	UPPSC, UPSSSC notifications	~85%	Hindi-first bilingual. Two-column Gazette layout. Dates buried in Hindi text — needs translation. More variation year to year.
RRB_GAZETTE	Railway Gazette scanned notifications	~78%	Always scanned. 300 DPI OCR. Two-column. Post-wise vacancy always in table. Age in numbered para 4 or 5.

Template matching is simple: when a new PDF arrives, check if its source\_id is in the template library. If yes, apply the source-specific parsing rules before generic parsing. The editor creates a new template entry every time they spend > 5 minutes correcting a record from a source they have not seen before. After 3 approved records from the same source using the same template, the template is considered trusted.

## Special Case: Corrigenda and Multi-PDF Notifications

Indian government notifications frequently issue corrections (corrigenda) after the original notification. Vacancy counts change, dates extend, eligibility rules are amended. The pipeline must handle these without corrupting already-approved records.

### Corrigendum detection:

- A corrigendum PDF is detected when: (a) it references a notification number that matches an existing approved record, or (b) its title contains words like "Corrigendum", "Amendment", "Addendum", "Erratum".
- Corrigenda are NEVER auto-merged into approved records. They always go to human review with a diff view showing what changed.
- Fields that corrigenda typically change: application\_end (extension), total\_vacancies (increase/decrease), exam\_date (postponement). These are tracked with a changes\_log JSONB field on the exam record.

### Multi-PDF notifications:

- Main notification PDF: Contains eligibility, age, dates → processed normally.
- Vacancy annexure PDF: Released separately, sometimes weeks later. Linked to main record via notification\_number. Triggers an update to the vacancies\_by\_category field once approved.

- Syllabus/scheme PDF: Not processed for exam tracker data — stored as a link only.

## Processing Queue Architecture

The pipeline runs as a background job queue, not in real-time. This prevents a slow OCR job from blocking fast text extractions and allows retry logic for failures.

Queue Name	Workers	SLA (target)	What goes here
pdf.acquisition	2 workers	< 30 seconds	All new PDF URLs from scraper. Download, hash, dedup. Fast queue.
pdf.classify	2 workers	< 1 minute	All downloaded PDFs pending classification. Very fast — just probing.
pdf.extract.text	4 workers	< 2 minutes	Type A/E PDFs (digital text). Fast pdfplumber extraction.
pdf.extract.ocr	2 workers	< 10 minutes	Type B/C/D PDFs (scanned). Slow Tesseract OCR. Separate queue prevents blocking.
pdf.parse	4 workers	< 3 minutes	All extracted text. Regex + Haiku calls. Haiku API latency is main factor.
pdf.review	Human	Target < 4 hrs	All records needing human review. Admin dashboard queue. Priority by application_end date.

All queues run on BullMQ (Node.js) backed by Upstash Redis. Each job has a maximum 3 retries with exponential backoff. Failed jobs after 3 retries go to a dead-letter queue and alert the engineering team.

### PRIORITY QUEUE RULE

If application\_end is within 7 days, the entire job chain for that PDF is prioritised. All stages run at maximum priority. Human review is flagged URGENT in the admin dashboard. The worst outcome is a PDF that takes 3 hours to process and misses notifying users about a 24-hour application window.

## Technology Stack for the Pipeline

Stage	Library / Tool	Language	Why this choice
Acquisition	<i>Node.js fetch + Supabase Storage</i>	TypeScript	Same runtime as main backend. Storage SDK already in project.
Classification (digital probe)	<i>pdfjs-dist</i>	TypeScript	Runs in Node.js — no Python subprocess for fast probe.
Classification (table detection)	<i>Camelot-py (probe only)</i>	Python	Best-in-class table detection. Quick lattice mode probe.
Extraction: digital text	<i>pdfplumber</i>	Python	Best word-level coordinate extraction. Preserves layout.
Extraction: OCR	<i>pytesseract + Tesseract 5</i>	Python	Tesseract 5 has best accuracy for eng+hin combined.
Extraction: tables	<i>Camelot (lattice mode)</i>	Python	Best handling of bordered government vacancy tables.
Parsing: regex layer	<i>Custom regex library</i>	TypeScript	Runs in main backend. No subprocess overhead for most cases.

Stage	Library / Tool	Language	Why this choice
<b>Parsing: LLM layer</b>	<i>Claude Haiku (claude-haiku-4-5)</i>	TypeScript / API	Fast, cheap, accurate for structured extraction tasks.
<b>Queue system</b>	<i>BullMQ + Upstash Redis</i>	TypeScript	Already in architecture. Serverless Redis — no infra to manage.
<b>Python worker runner</b>	<i>Railway.app background worker</i>	Python	Separate Python service for pdfplumber + Camelot + Tesseract.
<b>Admin review dashboard</b>	<i>Next.js + Supabase</i>	TypeScript	Part of main app. Protected route. Editor sees flagged records.

**NODE vs PYTHON** The main backend is TypeScript/Node. Python is used ONLY for pdfplumber, Camelot, and Tesseract — there is no good Node alternative for these. The Python worker runs as a separate Railway.app service. The Node backend calls it via a simple HTTP API endpoint: POST /extract with the PDF path, gets back extracted text and table data as JSON.

## MVP Build Strategy: What to Build First

Do not build all 7 stages at once. Build in this sequence to get to a working product fast while laying the right foundations.

Sprint	What you build	Stages covered	Why this order
Sprint 1	<b>100% manual entry</b>	<i>Stage 7 only (the admin form)</i>	Start by building the destination: the admin dashboard form that creates exam records. This defines the schema in practice. Manually enter the first 50 exams. Ship to users. Validate the schema is correct before writing any parser.
Sprint 2	<b>PDF download + storage</b>	<i>Stages 1–2</i>	Add acquisition and classification. The scraper now auto-downloads PDFs and stores them. Humans still do all parsing by reading the PDF and filling the form. But the PDF is now one click away in the admin dashboard.
Sprint 3	<b>Text extraction + regex parser</b>	<i>Stages 3–4 (Type A only)</i>	Add pdfplumber extraction and the regex rule library for digital text PDFs. This handles SSC, IBPS, SBI, UPSC — the P0 sources. Covers ~60% of all notifications. Skip OCR for now.
Sprint 4	<b>Confidence scoring + structured output</b>	<i>Stages 5–6</i>	Add the structuring and scoring layer. Now the admin form pre-fills with extracted values. Editor just reviews and approves. This is the productivity multiplier — 10x faster than manual entry.
Sprint 5	<b>OCR + template library</b>	<i>Stage 3 (Types B/C/D/E)</i>	Add Tesseract OCR for scanned PDFs (State PSCs, RRB Gazette). Build the first 10 templates for top sources. Confidence scores improve dramatically.

**THE RIGHT INSIGHT** The pipeline is a productivity tool for the human editor — not a replacement for them. Even at full automation, an editor spending 30 seconds approving a pre-filled record is 10x faster than spending 5 minutes filling it from scratch. The goal is never zero human involvement. The goal is making human review so fast that one editor can approve 100+ records per hour.