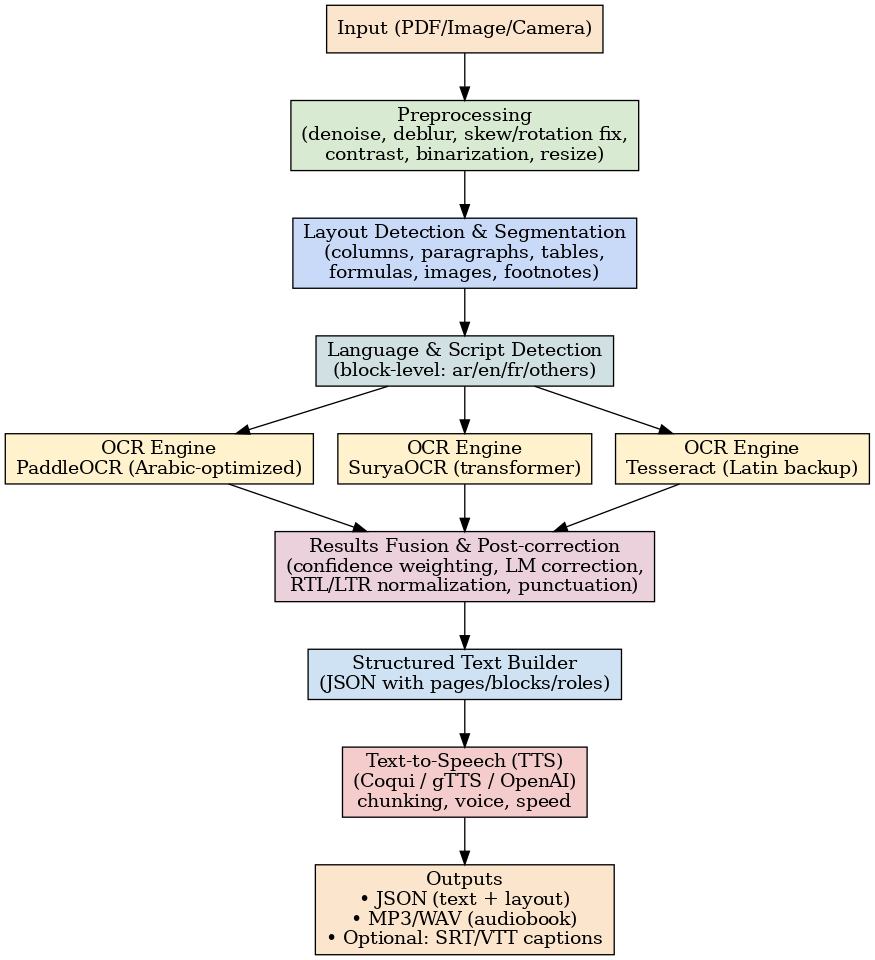
Arabic Accessibility OCR → Speech (Multi-OCR) — Project Plan

Goal: Convert scanned Arabic/multilingual documents (including noisy, multi-column pages) into accessible audio output while preserving reading order and providing structured text for captions.

# System Architecture



# Phases (Deep Details — Focused)

## Phase 1 — Setup & Research

Objectives: Establish a solid foundation with a modular repo, select core libraries, and define evaluation metrics and datasets.

Scope & Requirements:

• Input modalities: PDFs, images (PNG/JPG), optional live camera frames.

• Languages: Arabic (primary), English (secondary), optional French. Mixed-language documents expected.

• Accessibility outcome: High-quality TTS with natural prosody; JSON output for captions.

Tooling:

• Preprocessing: OpenCV, Pillow, pikepdf/PyMuPDF for PDF page rasterization.

• Layout: layoutparser + Detectron2 (or YOLOv8-seg) for block detection; fallback heuristics.

• OCR Factory: PaddleOCR (Arabic), SuryaOCR (transformer), Tesseract (Latin backup).

• Language Detection: fastText or langdetect; script heuristics (Arabic Unicode ranges).

• TTS: Coqui TTS (offline), gTTS (quick baseline), optional OpenAI TTS.

• Experiment tracking: MLflow; config via Hydra or pydantic settings.

Datasets & Benchmarks:

• Public Arabic OCR corpora; custom scans with multi-columns and stamps; create small validation set with GT text.

• Metrics: character/word accuracy; WER/CER; block detection mAP/F1; listening quality MOS-style user study.

Deliverables:

• Repo skeleton (src/, configs/, data/, tests/).

• Minimal E2E: PDF → single OCR engine → TTS MP3 (baseline).

## Phase 2 — Preprocessing & Layout Detection

Objectives: Improve OCR readiness; robustly segment pages into coherent blocks.

Preprocessing steps:

• Denoise (bilateral/median), deblur (unsharp), contrast (CLAHE), binarization (Otsu/SAUVOLA).

• Skew/rotation correction (Hough lines, deskew by text angle).

• Resize/scaling for OCR; normalize DPI; remove borders, stamps (optional masks).

Layout detection:

• Detect columns, paragraphs, tables, figures, formulas, footers/headers using layoutparser models.

• Reading order: sort by column flow (RTL for Arabic, LTR for Latin); detect nested regions.

• Page-wide vs block masks for OCR cropping; store bounding boxes with confidence.

Deliverables:

• JSON per page: list of blocks with bbox, type, confidence, reading order index.

• Visualizer script to overlay boxes for QA.

## Phase 3 — Multi-OCR Recognition & Language Routing

Objectives: Route each block to the most suitable OCR engine and fuse results for maximal accuracy.

Language & script detection:

• fastText/langdetect predictions; script heuristics (Arabic vs Latin numerals/symbols); model confidence thresholds.

Routing strategy:

• Arabic blocks → PaddleOCR (fine-tuned).

• Latin-heavy blocks (tables, codes) → Tesseract/SuryaOCR.

• Ambiguous blocks → run 2 engines → select by confidence/LM score.

Fusion & post-correction:

• Confidence-weighted voting; LM-based correction for Arabic diacritics, punctuation normalization.

• RTL/LTR normalization; join hyphenated lines; preserve numerals and math tokens.

Deliverables:

• Block-level transcripts with per-engine confidences.

• Script to compare engines on validation set; MLflow logs.

## Phase 4 — Structured Text & Prosody Prep

Objectives: Build clean hierarchical text and add hints for TTS prosody.

Structure builder:

• Assemble blocks into pages/sections with roles (heading, paragraph, list, caption).

• Normalize whitespace; fix sentence boundaries; handle bullet/numbered lists; keep table summaries.

Prosody cues for TTS:

• Insert pauses after headings; longer pauses between sections; keep punctuation consistent for natural speech.

• Optional: SSML layer for engines that support it (rate/pitch/break).

Deliverables:

• JSON output ready for TTS; optional Markdown for review.

• Unit tests for ordering and role assignment.

## Phase 5 — TTS Pipeline & Audiobook Generation

Objectives: Generate clear, continuous audio with configurable voice, speed, and chunking.

Chunking & batching:

• Split text into ~20–30s chunks; avoid mid-sentence cuts; maintain page/section markers.

• Cache repeated phrases (headers/footers).

TTS engines:

• Start with Coqui or gTTS; expose config for voice, rate, and sample rate; fallback paths if a chunk fails.

Post-processing:

• Normalize loudness (LUFS), remove leading/trailing silences, optional noise gate.

• Concatenate chunks; generate MP3/WAV; produce SRT/VTT captions aligned to chunks.

Deliverables:

• End-to-end: PDF → JSON → MP3 with captions.

• CLI command: ocr2speech input.pdf --lang auto --out book.mp3

## Phase 6 — Evaluation, Deployment & UX

Objectives: Validate quality with users; deploy as CLI/API; polish UX for accessibility.

Evaluation:

• OCR: CER/WER; layout F1; human QA on tricky pages (stamps, rotated tables).

• TTS: MOS-like listening tests; intelligibility at various speeds; Arabic-specific pronunciation checks.

Deployment:

• Package as Python package; FastAPI service with upload; Docker image for reproducibility.

• Configurable resource usage (GPU/CPU); offline/online modes; caching to control costs.

Accessibility & UX:

• Keyboard-first CLI; screen-reader-friendly API docs; options for speaking rate and voice selection.

• Optional desktop app with big controls; hotkeys to skip sections.

Deliverables:

• Dockerized FastAPI + CLI; sample datasets; demo video/script; user guide.