Industrial Project 2026-07-10

No cover image available.

AI-Powered Telecom Copper Reclamation: Document Intelligence and Workflow Automation for ACR/CAPR/CPR Analysis

A decision-support and workflow-modernization platform for telecom copper reclamation that parses ACR, CAPR, and CPR reports, normalizes cable-pair evidence, generates conservative review recommendations, models With-SOW and Without-SOW processes, and provides a FastAPI and React/Vite foundation for governed automation.

Project Snapshot

DATE
2026-07-10

CATEGORY
Industrial Project

SHORT DESCRIPTION
A decision-support and workflow-modernization platform for telecom copper reclamation that parses ACR, CAPR, and CPR reports, normalizes cable-pair evidence, generates conservative review recommendations, models With-SOW and Without-SOW processes, and provides a FastAPI and React/Vite foundation for governed automation.

TAGS / TECH STACK

Telecom | Network Automation | Copper Reclamation | Document Intelligence | Workflow Automation | Decision Support

Project Links

GitHub Link | GitHub Link | GitHub Link | GitHub Link | GitHub Link
GitHub Link | GitHub Link | GitHub Link | GitHub Link | GitHub Link
GitHub Link | GitHub Link | GitHub Link | Project Link | Project Link
GitHub Link | Project Link | GitHub Link | Project Link | Project Link
Project Link | Project Link | GitHub Link | GitHub Link | GitHub Link
GitHub Link | GitHub Link | GitHub Link

Full Project Content

AI-Powered Telecom Copper Reclamation Workflow Automation Platform is a decision-support and workflow-modernization prototype for analyzing ACR, CAPR, and CPR reports, classifying cable-pair evidence, generating conservative reclamation recommendations, mapping engineering workflows, and identifying opportunities for governed automation.

View the GitHub Repository

GitHub Repository: dranubhaparashar/AI-Powered-Telecom-Copper-Reclamation-Workflow-Automation-Platform

---

Wiki documentation: Home · About the Platform · Architecture and Design · ACR/CAPR/CPR Parsing · Decision Logic

Application documentation: Workflow Variants · Dashboard and Frontend · API Reference · Installation and Local Run

> **Engineering and governance:** Testing and Validation · Security, Privacy, and Claim Boundary · Known Issues and Roadmap

---

One-Line Idea

Transform fragmented telecom reclamation evidence and multi-stage engineering processes into a structured, explainable, reviewable workflow—without allowing automated parsing to replace authoritative engineering, customer-impact, billing, field, safety, permit, or operational approvals.

Why This Project Exists

Telecom copper-reclamation work is not a single document-processing task. It is a cross-functional decision process involving engineering reports, cable and pair records, circuit evidence, planning information, field observations, design packages, quality-control steps, permits, O-Calc activities, customer-impact checks, and operational approvals.

A large part of this work is still coordinated through PDF reports, spreadsheets, emails, manual lookups, and repeated handoffs. Engineers may need to:

- locate and interpret ACR, CAPR, and CPR reports;
- identify cable, count, terminal, circuit, and pair-level information;
- determine whether a pair appears working, assigned, defective, spare, or unresolved;
- reconcile IN COUNT, OUT COUNT, and QUALIFIED PAIRS relationships;
- consolidate evidence into review workbooks;
- compare With-SOW and Without-SOW operating processes;
- estimate where automation could reduce repetitive work;
- retain enough evidence for engineering review and audit.

This project organizes those activities into two complementary layers:

1. **Document intelligence and decision support** for extracting and normalizing evidence from ACR/CAPR/CPR reports.
2. **Workflow modernization** for exposing process data through FastAPI and presenting workflow phases, automation opportunities, and savings scenarios in a React/Vite dashboard.

The goal is not automatic decommission authorization. The goal is to reduce avoidable manual effort, improve consistency, make decision reasons visible, and create a foundation for a governed human-review workflow.

Project at a Glance

Area	Description
Project type	Industrial decision-support and workflow-automation prototype
Domain	Telecom copper reclamation and decommission planning
Primary inputs	ACR, CAPR, and CPR PDF reports; process-flow configuration
Document-processing output	Parsed metadata, pair records, count relationships, normalized statuses, warnings, and preliminary recommendation
Workflow output	With-SOW and Without-SOW process views, phase durations, automation opportunities, and savings scenarios
Backend	Python, FastAPI, Pydantic, Uvicorn
Frontend	React, TypeScript, Vite, Tailwind CSS, Framer Motion
Parsing	PyMuPDF and deterministic Python rules
Data processing	pandas, CSV, JSON, XLSX
Main users	Telecom engineering teams, planning teams, field operations, design/QC teams, workflow analysts, and automation engineers
Current maturity	Documented prototype requiring further integration, validation, security controls, and production hardening
Safety boundary	Decision support only; human engineering and field validation remain mandatory

What Makes the Platform Different

The project treats copper reclamation as both an **evidence problem** and a **workflow problem**.

Conventional approach	Platform approach
Read each report manually	Extract report metadata and pair-level evidence into structured records
Interpret status codes independently	Normalize known statuses and preserve raw evidence

Conventional approach	Platform approach
Maintain findings in disconnected spreadsheets	Produce repeatable CSV, JSON, and XLSX outputs
Provide a binary recommendation without explanation	Return conservative decision categories with reason codes and warnings
Compare workflows informally	Model With-SOW and Without-SOW process variants as structured data
Estimate savings without process transparency	Tie savings scenarios to documented activities and configurable assumptions
Treat missing parsed evidence as absent	Convert uncertain or unparsed evidence into warnings and manual review
Allow automation to imply operational approval	Explicitly separate decision support from decommission authorization

> **Important:** The repository name uses “AI-Powered,” but the inspected document-analysis implementation is currently deterministic and rule-based. It is not yet a trained or independently validated machine-learning model.

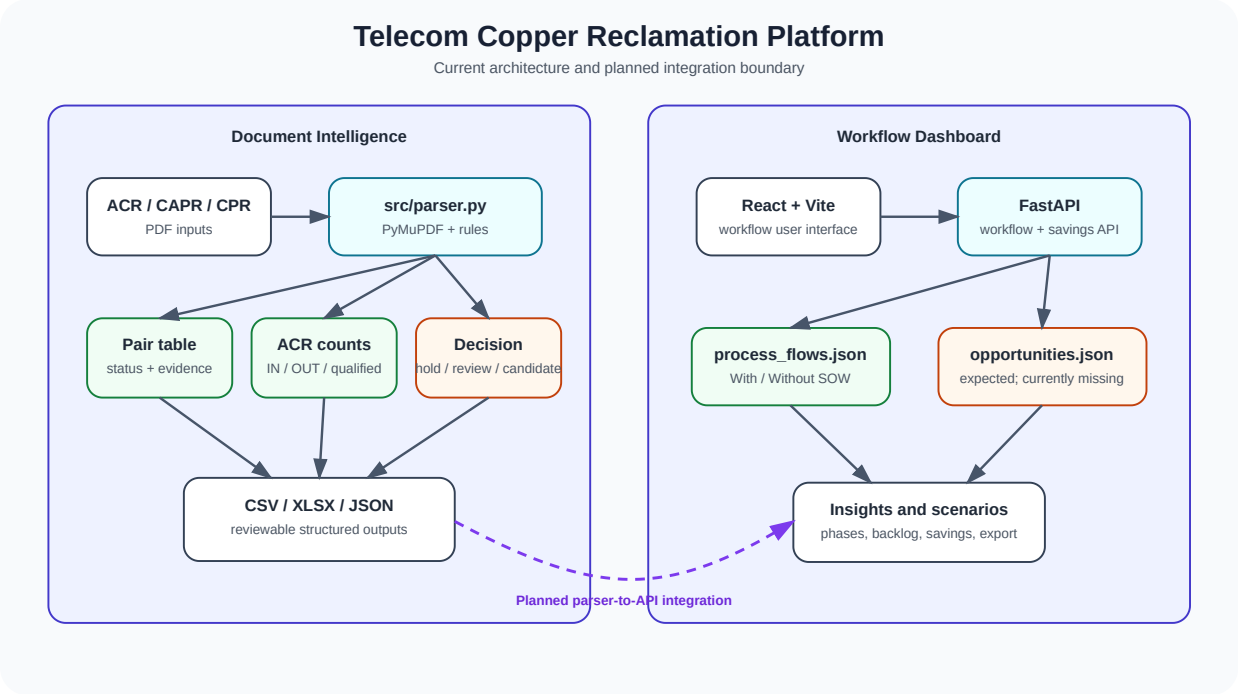
Reader Walkthrough

A reader can understand the platform in eight steps:

1. ACR, CAPR, and CPR PDF reports are collected for a candidate reclamation project.
2. The parser extracts text with PyMuPDF and detects the report family.
3. Metadata, cable/count sections, pair rows, statuses, defects, and available circuit identifiers are parsed.
4. Raw statuses are normalized into working, review-required, defective, spare, or unknown categories.
5. Evidence is aggregated into a conservative recommendation: **HOLD**, **ENGINEERING REVIEW REQUIRED**, or **PROCEED CANDIDATE**.
6. Structured results are exported for engineering review in CSV, JSON, or XLSX form.
7. The FastAPI and React layers represent the broader reclamation workflow, including With-SOW and Without-SOW process variants.

8. Human reviewers verify all authoritative systems, field conditions, customer impact, permits, design dependencies, and approved work procedures before any operational action.

Platform Architecture



The architecture currently contains two main subsystems.

1. Document Intelligence

The document-intelligence layer receives ACR/CAPR/CPR PDFs, parses report content, produces pair-level and count-level tables, and generates a project summary with warnings and preliminary decision logic.

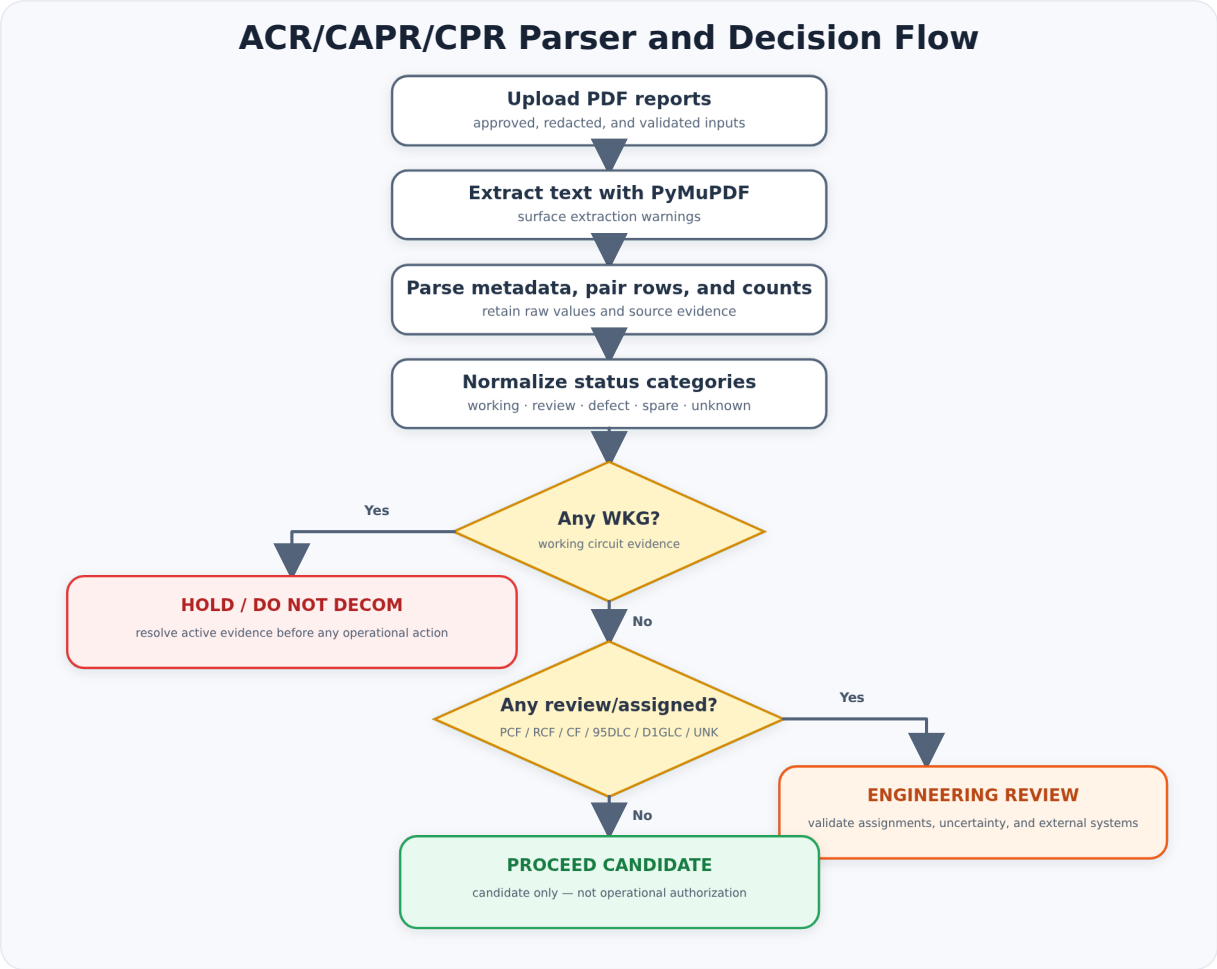
2. Workflow Application

The workflow layer uses a FastAPI backend and React/Vite frontend to serve and visualize process variants, phase details, automation opportunities, and savings calculations.

The current repository documents a **planned integration** between parser outputs and the API/dashboard. This connection is a key roadmap item rather than a completed production feature.

Read the Architecture and Design Wiki page

Document Intelligence Pipeline



The parser follows a structured sequence:

1. Read PDF text.
2. Detect whether the document resembles an ACR, CAPR, or CPR report.
3. Extract report-level metadata.
4. Parse pair-level records and cable/count sections.
5. Preserve the original raw status.
6. Normalize the status for downstream interpretation.
7. Aggregate working, assigned/review, defective, spare, and unknown evidence.
8. Generate a preliminary recommendation and warnings.
9. Export structured review files.

Supported Report Families

Report family	Intended evidence
ACR	Cable/count relationships, pair information, IN COUNT, OUT COUNT, and QUALIFIED PAIRS sections

Report family	Intended evidence
CAPR	Cable and pair assignment details with related references
CPR	Cable/pair status and available service or circuit evidence

Exact report layouts can vary. A production-grade implementation must maintain layout-specific test fixtures and should never treat “not parsed” as “not present.”

Read the ACR/CAPR/CPR Parsing Wiki page

Extracted Information

Depending on the source report and available text, the parser is designed to organize fields such as:

- source filename and report family;
- report date;
- work center and employee identifier;
- cable identifier and pair range;
- pair number;
- raw pair status;
- normalized status;
- circuit or service identifiers;
- line and pair state;
- binding post and color;
- terminal and location information;
- defect indicators;
- IN COUNT, OUT COUNT, and QUALIFIED PAIRS relationships;
- file-level warnings and recommendation.

For production traceability, every extracted record should additionally retain source page, evidence text, source-file hash, parser version, extraction timestamp, quality flags, and reviewer disposition.

Read the Data Model and Exports Wiki page

Pair-Status Interpretation

The current rule set uses conservative status categories.

Category	Current values	Interpretation	Preliminary handling
Working	`WKG`	Active or working evidence	Hold; do not decommission yet
Review	`PCF`, `RCF`, `CF`, `95DLC`, `D1GLC`, `UNK`	Assigned, unresolved, or review-required evidence	Engineering review required
Defective	`DEF`	Defect evidence	Retain and validate disposition
Spare	`SPR`, `SPARE`	Spare indication	Potential candidate only after validation
Unknown	Blank or unrecognized	Insufficiently classified evidence	Manual investigation required

Raw source values must be retained even after normalization. That allows reviewers to inspect exactly what was present in the source report and prevents the normalized label from hiding parsing ambiguity.

Conservative Decision Logic

The platform deliberately avoids turning report parsing into automatic authorization.

HOLD / DO NOT DECOM YET

Returned when working evidence such as `WKG` is detected. The result indicates that active evidence must be resolved through authoritative validation.

ENGINEERING REVIEW REQUIRED

Returned when assigned, ambiguous, unresolved, or review-status evidence is present. The project cannot be automatically cleared from the parsed inputs.

PROCEED CANDIDATE

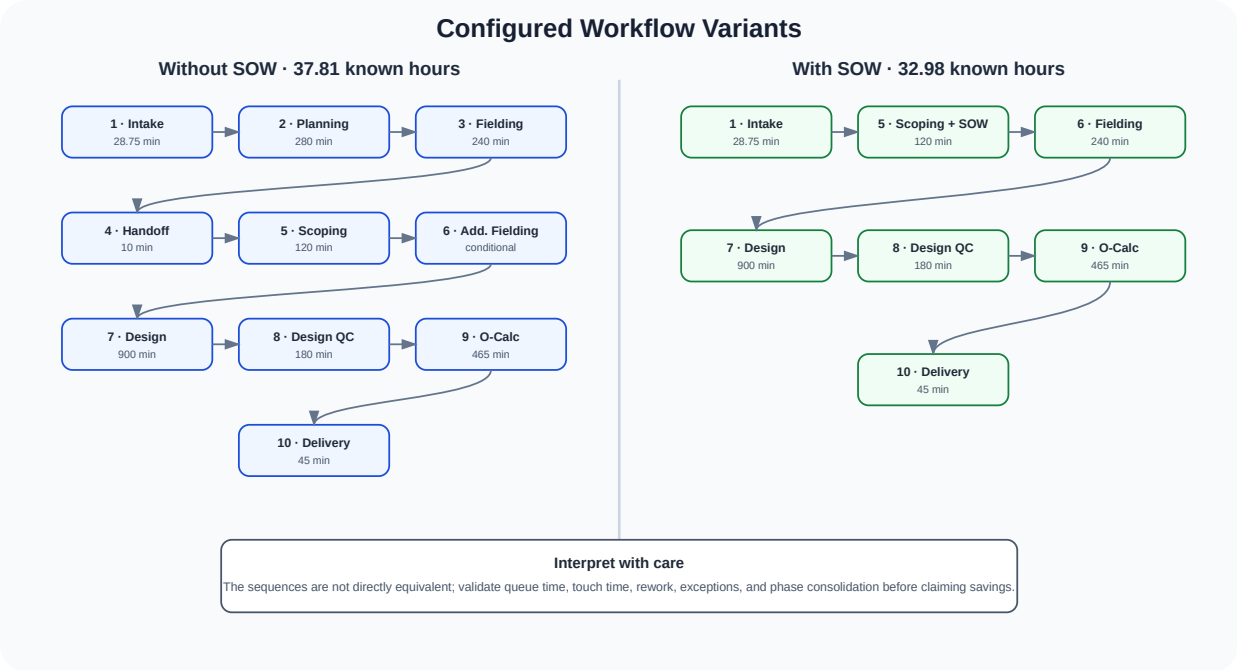
Returned only when the parsed evidence does not contain a working or review-required status.

> **A PROCEED CANDIDATE result does not prove that the plant is safe to remove.** It only means that the parsed reports did not reveal working or review-status evidence under the implemented rules.

Before operational approval, teams must validate live circuit and service records, billing/customer state, pending orders or migrations, field labels and conditions, network topology and design dependencies, environmental and safety constraints, permits, right-of-way requirements, and the approved work package.

Read the complete Decision Logic Wiki page

Workflow Variants



The backend contains two structured process-flow variants.

Without SOW

The current sequence includes:

1. Intake
2. Planning
3. Fielding
4. Design Handoff
5. Scoping/Pre-Design
6. Conditional Additional Fielding
7. Design
8. Design QC
9. O-Calc
10. Delivery

The known-duration activities total approximately **2,268.75 minutes, or 37.81 hours**, excluding conditional additional fielding.

With SOW

The current sequence includes:

1. Intake
2. Scoping with SOW
3. Fielding
4. Design
5. Design QC
6. O-Calc
7. Delivery

The known-duration activities total approximately **1,978.75 minutes, or 32.98 hours**.

The raw difference is approximately **290 minutes, or 4.83 hours**. However, the sequences are not directly equivalent, and some activities are omitted, consolidated, or conditioned differently. These configured values must not be presented as proven business savings until they are calibrated with controlled operational measurements.

A proper validation study should separately measure queue time, touch time, rework, handoffs, exception rates, systems accessed, role-based loaded cost, completion quality, and comparability between workflow variants.

Read the Workflow Variants Wiki page

FastAPI Service

The backend exposes workflow and savings-related data through FastAPI.

Method	Endpoint	Purpose
`GET`	`/health`	Basic service health
`GET`	`/api/flows`	Return both workflow variants
`GET`	`/api/flows/{variant}`	Return `with_sow` or `without_sow`
`GET`	`/api/summary`	Return platform and workflow summary
`GET`	`/api/automation-opportunities`	Return the automation backlog
`POST`	`/api/savings`	Estimate time and cost savings from selected assumptions

Example savings request:

```
{
  "selected_opportunity_ids": ["opp-1"],
  "automation_coverage_pct": 50,
  "jobs_per_month": 20,
  "work_hours_per_day": 8,
  "loaded_hourly_rate": 75
}
```

The current documented snapshot notes that some endpoints depend on `backend/app/data/automation_opportunities.json`, which was not visible in the inspected backend data directory.

Read the API Reference Wiki page

React/Vite Dashboard

The frontend is designed as an operational visualization layer for:

- comparing With-SOW and Without-SOW processes;
- viewing phase-level sequence and duration;
- displaying workflow summaries and pain points;
- exploring automation opportunities;
- calculating scenario-based time and cost savings;
- presenting a clearer process narrative to engineering and operations stakeholders.

The application uses React, TypeScript, Vite, Tailwind CSS, Framer Motion, and Lucide React.

The repository documentation identifies two imported frontend components—`AutomationBacklog` and `SavingsCalculator`—that were not visible in the inspected component directory. These should be restored or replaced before the frontend is described as a clean end-to-end build.

Read the Dashboard and Frontend Wiki page

Generated Outputs

The parser can support review-oriented outputs such as:

- pair-level CSV;
- ACR count-section CSV;

- project or file decision-summary JSON;
- uploaded-file summary tables;
- XLSX analysis workbooks;
- warnings and reason-code summaries.

Recommended production fields include:

- source filename and cryptographic hash;
- source page and evidence span;
- report type and report date;
- raw and normalized status;
- cable, count, pair, terminal, and circuit identifiers;
- extraction timestamp and parser version;
- validation or quality flags;
- recommendation reason codes;
- reviewer name, review date, disposition, and approval history.

Exports should preserve raw evidence, prevent spreadsheet-formula injection where relevant, classify or watermark sensitive data, and avoid overwriting approved historical records.

Technology Stack

Layer	Technologies
PDF processing	Python, PyMuPDF (`fitz`)
Structured processing	pandas, JSON, CSV, XLSX
Backend API	FastAPI, Pydantic, Uvicorn
Frontend	React, TypeScript, Vite
UI	Tailwind CSS, Framer Motion, Lucide React
Testing	pytest
Version control	Git and GitHub
Documentation	README and 16-page GitHub Wiki

Repository Structure

```
AI-Powered-Telecom-Copper-Reclamation-Workflow-Automation-Platform/
├── assets/
│   └── diagrams/
│       ├── platform_architecture.svg
│       ├── parser_decision_flow.svg
│       └── workflow_variants.svg
├── backend/
│   ├── app/
│   │   ├── data/
│   │   │   └── process_flows.json
│   │   ├── main.py
│   │   └── models.py
│   └── requirements.txt
├── data/
├── docs/
│   └── DASHBOARD_DETAILS.md
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   ├── App.tsx
│   │   ├── api.ts
│   │   └── types.ts
│   └── package.json
├── outputs/
├── src/
│   └── parser.py
├── tests/
├── parsed_acr_output.xlsx
├── telecom_analysis.xlsx
└── README.md
```

Read the Repository Structure Wiki page

Local Run

Run the PDF Parser

Create and activate a Python environment from the repository root:

```
python -m venv .venv
```

Windows PowerShell:

```
.\.venv\Scripts\Activate.ps1
```

```
pip install pandas pymupdf openpyxl pytest
```

macOS/Linux:

```
source .venv/bin/activate
pip install pandas pymupdf openpyxl pytest
```

The parser is currently exposed as a Python library rather than a complete command-line application.

Run the FastAPI Backend

```
cd backend
python -m venv .venv
```

Windows PowerShell:

```
.\.venv\Scripts\Activate.ps1
pip install -r requirements.txt
uvicorn app.main:app --reload --port 8000
```

macOS/Linux:

```
source .venv/bin/activate
pip install -r requirements.txt
uvicorn app.main:app --reload --port 8000
```

Useful local URLs:

- `http://localhost:8000/health`
- `http://localhost:8000/docs`
- `http://localhost:8000/api/flows`

Run the Frontend

```
cd frontend
npm install
npm run dev
```

For a custom API URL, create `frontend/.env.local`:

```
VITE_API_BASE_URL=http://localhost:8000
```

Read Installation and Local Run

Current Implementation Status

Capability	Status
ACR/CAPR/CPR text extraction	Implemented
Pair/status parsing	Implemented using deterministic rules
ACR count parsing	Implemented
CSV/JSON/XLSX examples	Present
Process-flow API	Implemented
React/Vite dashboard shell	Implemented
Parser-to-dashboard integration	Planned
Trained and evaluated AI/ML model	Not present in the inspected implementation
Production validation and authorization workflow	Not complete
Secure enterprise identity and audit controls	Not visible in the current prototype

---

Known Issues

The current Wiki documents the following visible gaps:

1. `automation_opportunities.json` is referenced by the API but was not visible in the backend data directory.
2. `AutomationBacklog` and `SavingsCalculator` are imported by `App.tsx` but were not visible in the frontend component directory.
3. Parser dependencies are not declared in the backend requirements file.
4. Parser outputs are not connected to the API and UI.
5. Tests do not yet validate the complete business behavior.
6. Authentication, authorization, secure storage, and audit workflow are not yet visible.
7. No trained and evaluated AI model is present in the inspected implementation.
8. Sample PDF and XLSX files require privacy and approval review.
9. No license file was visible in the documented snapshot.

These limitations are included to keep the portfolio description technically credible and to distinguish the working prototype from the intended production platform.

Read Known Issues and Roadmap

Roadmap

Phase 1 — Reproducible Prototype

- restore missing backend data and frontend components;
- add root-level dependency management;
- ensure backend and frontend clean builds;
- add configuration validation and graceful error handling;
- introduce continuous integration.

Phase 2 — Tested Document Intelligence

- create synthetic and approved redacted fixtures;
- add golden-file parser tests;
- retain page-level evidence;
- version parser rules;
- measure extraction quality and document-layout coverage;
- add OCR fallback only where justified and evaluated.

Phase 3 — Integrated Review Workflow

- connect secure uploads to parser jobs;
- expose parser evidence through the API;
- add reviewer reason codes and disposition;
- retain a tamper-evident audit history;
- implement governed exports and role-based access.

Phase 4 — Enterprise Integration

- integrate authoritative circuit, billing, GIS, field, design, and permit systems;
- introduce event-driven workflow orchestration;
- add monitoring, observability, and operational support;
- measure cycle-time, rework, quality, and customer-impact outcomes.

Security, Privacy, and Claim Boundary

Telecom engineering documents may contain operationally sensitive information, customer or circuit identifiers, employee identifiers, addresses, facility information, and network-topology details.

Minimum safeguards for any real deployment include:

- synthetic or formally approved redacted data in public repositories;
- encryption in transit and at rest;
- role-based access control;
- strong authentication and authorization;
- secrets outside source control;
- sensitive-field redaction in logs and screenshots;
- retention and deletion rules;
- auditable reviewer and approval history;
- secure export controls;
- human approval before any operational action.

The Project Does Claim To

- structure ACR/CAPR/CPR evidence into reviewable records;
- implement deterministic status normalization and conservative decision support;
- model two reclamation workflow variants;
- provide an API and dashboard foundation;
- document known limitations and a production roadmap.

The Project Does Not Claim To

- authorize copper removal or decommissioning;
- prove that a cable is free of live customer service;
- replace billing, circuit, GIS, field, design, permit, or safety systems;
- provide a fully integrated production platform;
- provide a trained and independently validated AI model;
- provide legal, regulatory, safety, or engineering certification.

Read Security, Privacy, and Claim Boundary

Testing and Validation Strategy

The current tests are starter/import checks. A production validation program should add:

- synthetic and approved redacted document fixtures;
- report-layout-specific unit tests;
- golden-file regression tests;
- malformed, scanned, rotated, and partially readable PDF cases;
- status-normalization boundary tests;
- count-relationship validation tests;
- decision-logic tests;

- FastAPI success and error tests;
- frontend component and end-to-end tests;
- parser-to-dashboard contract tests;
- security, authorization, and audit tests;
- operational validation against qualified engineering review.

Key measures should include field-level precision and recall, row-level extraction accuracy, unparsed-row rate, false-clear rate, document coverage, reviewer agreement, cycle time, rework rate, and exception rate.

Read Testing and Validation

Complete Wiki Documentation

Wiki page	Purpose
Wiki Home	Project entry point, safety boundary, maturity, and navigation
About the Platform	Problem statement, users, scope, and value
Architecture and Design	System layers, data flow, and integration boundaries
ACR/CAPR/CPR Parsing	Supported report families, extraction steps, fields, and limitations
Decision Logic	Status categories, recommendations, and mandatory external checks
Data Model and Exports	Parser outputs, production fields, and export safety
Workflow Variants	With-SOW and Without-SOW sequences, durations, and calibration
Dashboard and Frontend	React/Vite application structure and UI responsibilities
API Reference	FastAPI endpoints, accepted variants, and savings request
Installation and Local Run	Parser, backend, and frontend setup
Testing and Validation	Test layers and production validation requirements
Repository Structure	Folder and file guide

Wiki page	Purpose
Developer Guide	Development, parser-rule, API, frontend, and release checklists
Known Issues and Roadmap	Current gaps and phased evolution plan
Security, Privacy, and Claim Boundary	Sensitive-data controls and explicit claim limits
Quick Links	Fast navigation across all documentation areas

Practical Use Cases

User group	How the platform helps
Telecom engineering	Structures report evidence and highlights working or unresolved pair conditions
Copper-reclamation teams	Provides a consistent preliminary review package
Field operations	Identifies evidence and questions that require field validation
Planning and design	Makes workflow phases and dependencies visible
Quality-control teams	Supports repeatable review fields and reason codes
Automation teams	Identifies integration points and repetitive process steps
Product and process analysts	Compares workflow variants and configurable savings scenarios
Technical reviewers	Provides transparent limitations, diagrams, Wiki pages, and roadmap
Data-governance teams	Establishes provenance, export, privacy, and audit requirements

---

Key Strengths

- Combines document extraction with workflow modernization.
- Preserves raw source evidence alongside normalized values.
- Uses conservative decisions instead of unsafe automatic clearance.
- Makes the difference between parsing and authorization explicit.

- Documents With-SOW and Without-SOW operating processes.
- Includes a FastAPI backend and React/Vite dashboard foundation.
- Provides structured export examples.
- Includes architecture diagrams and a 16-page GitHub Wiki.
- States implementation gaps and claim boundaries openly.
- Provides a credible path from prototype to governed enterprise platform.

Key Innovation

> The strongest contribution is not an unsupported claim of fully autonomous AI. It is the combination of evidence extraction, conservative decision logic, workflow modeling, explainable outputs, and explicit human-approval boundaries in one documented telecom-reclamation prototype.

The system recognizes that missing parsed evidence is not proof of absence and that a report-based recommendation cannot replace authoritative system and field validation. This is critical in infrastructure workflows where an incorrect “clear” decision could affect active service, customer operations, safety, compliance, or network integrity.

Project Links

- GitHub Repository
- GitHub Wiki
- README
- Source Parser
- Backend
- Frontend
- Architecture and Design
- Decision Logic
- API Reference
- Known Issues and Roadmap

Conclusion

The AI-Powered Telecom Copper Reclamation Workflow Automation Platform demonstrates how document intelligence and process automation can be combined to improve a complex engineering workflow.

The parser converts ACR, CAPR, and CPR report content into structured evidence. Conservative decision logic helps distinguish working, unresolved, defective, spare, and

unknown pair conditions. The workflow application models two operating variants and creates a foundation for API-driven dashboards, automation backlogs, and scenario-based savings analysis.

The project is most credible when understood as a **decision-support prototype and workflow foundation**. Its future value depends on rigorous document testing, parser-to-dashboard integration, governed enterprise APIs, strong data protection, measurable operational validation, and mandatory human approval.

Final Thought

> Extract the evidence. Preserve the source. Explain the decision. Never confuse a candidate recommendation with authorization.
>
> ACR/CAPR/CPR parsing · conservative pair-status logic · FastAPI · React/Vite · With-SOW and Without-SOW workflows · governed human review · complete GitHub Wiki

Source: src/content/posts/AI-Powered-Telecom-Reclamation-Complete-Website-Post.md