AI-Assisted Digitization of PDF Forms

A Technical Architecture for Intelligent Form Extraction

Executive Summary

The U.S. Chamber of Commerce reports that the federal government uses nearly 10,000 unique forms, and processes over 106 billion pieces of paperwork annually. Typically standardized as PDF files and paper formats, these forms present a significant challenge for organizations undergoing digitization initiatives: transforming them into interactive web applications that traditionally requires weeks of manual development work per form, depending on length and complexity. In many cases, digital/web forms exist in parallel with paper/PDF forms to offer users flexibility and accommodate various technology needs.

CADMUS LOGIC.AI℠

When policy mandates revisions to the paper or PDF forms, the digital forms need to stay up to date with their paper twin to ensure parity in data collection. We often see this with our federal government and large business customers that need to undergo regular updates to paper/PDF forms, at times with complicated business rule changes, which creates a challenging process for updating the digital versions.

This whitepaper presents an AI-assisted approach to form digitization as part of our Cadmus Logic.AI℠ ReGenX product suite designed to accelerate enterprise-scale modernization. Using large language models (LLMs) combined with intelligent post-processing, the ReGenX form digitization engine can extract complete form definitions from PDF documents in minutes rather than weeks. The system produces three distinct schema outputs that any front-end framework can consume directly to render fully functional, validated forms.

Background & Our Foundational Approach

Before exploring the AI-assisted approach, it's important to understand where we started. Since the late 2000s, we have been tinkering with schema-driven form conversion engines. We wrote thousands of lines of code that parsed PDF documents at the binary level, extracted field definitions, and produced schemas that could be used to build the digital forms. These are the exact same types of schemas that our AI system produces today.

We developed several iterations of pre-AI form conversion systems with significant engineering efforts. Those systems used tools like QPDF to decrypt and parse PDF internals, walked through page hierarchies and annotation objects, extracted field types from PDF form specifications, and assembled everything into structured JSON schemas. They mapped PDF field types to user interface (UI) components, extracted labels from text positioning data, and built complete form definitions programmatically.

Why This Foundation Mattered

Years of building this system through core engineering work taught us exactly what "good" output looks like. We defined the precise structure of a form definition schema, established how validation rules should be organized, and determined how flow-control logic should be expressed. In other words, we already had the formula for form digitization.

> "AI did not replace our domain expertise—it amplified it. We had already spent years hand-crafting and perfecting the output format through production use.

When we introduced AI into the process, we were not asking it to invent a schema or make architectural decisions. We were asking it to produce output that conforms to a proven, well-defined structure. The three output schemas that drive our transformation pipeline were not designed for the AI; they were created through years of iterative refinement, validated against real-world requirements, and only then taught to the AI as explicit extraction targets. In other words, we had already cracked the formula for form digitization. The remaining challenges were speed and accessibility—reducing weeks of development to minutes and enabling non-technical users to achieve the same results through natural language prompts rather than code. AI technology delivered on both fronts: it collapsed the timeline dramatically while putting the capability directly in the hands of people who understand forms, not just people who understand code.

AI did not replace our domain expertise—it amplified it. We had already spent years hand-crafting and perfecting the output format through production use. AI simply became a faster and more adaptable way to generate the same high-quality results. This is why the ReGenX form digitization process works

so effectively: the target was never ambiguous. We knew exactly what correct output looked like long before AI entered the equation.

The Core Innovation

This whitepaper describes an architectural approach to form digitization. The techniques are implementation-agnostic and can be adapted to different AI models and use cases.

Rather than treating AI as a black box that magically produces output, this architecture treats AI as an intelligent extraction layer that works within a deterministic pipeline. The AI understands document structure and semantics. Traditional code handles transformation, validation, and schema generation. Together, they produce consistent, production-ready output.

The Challenge

To demonstrate ReGenX's form digitization capabilities, we have applied it on high-volume immigration forms from the US Citizenship and Immigration Services (USCIS) like the I-485 form - Application to Register Permanent Residence. At 24 pages, it contains 14 distinct parts, dozens of sections, and hundreds of individual fields. Each field has specific requirements: data types, validation rules, conditional visibility logic, and relationships to other fields.

Based on our experience, a traditional digitization effort of such a form would require:

- Manual field-by-field HTML/component creation (40-60 hours)
- Writing individual validation rules (20-30 hours)
- Implementing conditional show/hide logic (15-20 hours)
- Testing and quality assurance (20-30 hours)
- Total: 100-140 hours of skilled developer time per form

When forms are updated (which happens regularly with government documents), the entire process must be repeated. This creates an unsustainable burden for organizations that work with many form types and versions.

The Solution Architecture

Our system processes PDF forms through a seven-step pipeline, producing three schema outputs that together provide everything needed to render and validate the form in any UI framework.

Processing Pipeline

Step 1: PDF Decryption Government PDFs often have security settings that prevent text extraction. The system automatically detects and removes these restrictions while preserving the document content.

Step 2: PDF Analysis Before extraction begins, the system analyzes the document structure. It detects the number of parts, estimates complexity, and determines whether single-pass or multi-pass extraction is optimal. For forms with more than 10 parts or 20 pages, multi-pass extraction is triggered automatically.

Step 3: AI-Powered Extraction The PDF is sent to an LLM with detailed instructions about what to extract and how to structure the output. The AI reads the entire document, understanding context, hierarchy, and relationships between elements.

Step 4: Schema Validation and Fixing The extracted data is validated against JSON schemas. Common extraction errors are automatically corrected. Missing required fields are flagged for review.

Step 5: Cross-Validation Field references are cross-checked to ensure that conditional logic refers to actual field IDs, and that all relationships are internally consistent.

Step 6: Flat Structure Transformation The hierarchical form definition is transformed into a flat array suitable for direct UI rendering, with each element containing all metadata needed for display.

Step 7: Validation Schema Generation AJV-compliant validation schemas are generated automatically, including conditional validation rules and custom error messages.

The Three Output Schemas

The system produces three distinct schemas, each serving a specific purpose in the form rendering and validation lifecycle.

1. Form Definition Schema (formDef)

This schema contains every field that will appear in the rendered form, with complete metadata for each element.

formDef Schema Structure

```
{
  "formId": "form-i-485",
  "formTitle": "Application to Register Permanent Residence",
  "formVersion": "Edition 07/15/24",
  "elements": [
    {
      "id": "cmpCurrentLegalName",
      "component": "fullname",
      "label": "Your Current Legal Name",
      "part_title": "Part 1. Information About You",
      "section_title": "1|",
      "required": true
    },
    {
      "id": "cmpMailingAddress",
      "component": "address",
      "label": "Mailing Address",
      "part_title": "Part 1. Information About You",
      "section_title": "2|Mailing Address",
      "required": true
    }
  ],
  "partTitles": ["Part 1. Information About You", ...],
  "sectionTitles": { "Part 1...": ["1|", "2|Mailing Address", ...] }
}
```

Key aspects of the form definition schema:

- Composite components (fullname, address) are automatically identified and prefixed with 'cmp' for easy UI component mapping
- Part and section titles are preserved for navigation and grouping
- Every element contains the metadata needed for rendering without additional lookups
- The flat structure allows simple iteration for rendering while maintaining hierarchical context

2. Validation Schema (validationSchema)

This AJV-compliant schema provides complete validation logic, including conditional requirements.

validationSchema Structure

```
{
  "type": "object",
  "properties": {
    "cmpCurrentLegalName": {
      "type": "object",
      "properties": {
        "givenName": {
          "type": "string",
          "minLength": 1,
          "pattern": "^[a-zA-Z0-9\\-']*$",
          "errorMessage": {
            "minLength": "This field is required.",
            "pattern": "Cannot include special characters except - and '."
          }
        },
        "lastName": {
          "type": "string",
          "minLength": 1,
          "errorMessage": { "minLength": "This field is required." }
        },
        "middleName": { "type": "string" }
      },
      "required": ["givenName", "lastName"]
    }
  },
  "allOf": [
    {
      "if": {
        "properties": { "has-other-names": { "const": "Yes" } },
        "required": ["has-other-names"]
      },
      "then": {
        "required": ["cmpOtherNames"]
      }
    }
  ]
}
```

The validation schema includes:

- Type validation for every field (string, number, array, object)
- Pattern validation for formatted fields (SSN, A-Number, dates, phone numbers)
- Conditional requirements using JSON Schema if/then/else constructs
- Custom error messages for each validation rule
- Support for composite field validation with nested required fields

3. Flow Control Schema (flowControl)

This schema defines conditional visibility—which fields should appear based on the values of other fields.

flowControl Schema Structure

```
[
  {
    "controlElements": ["has-other-names"],
    "requiredWhen": {
      "or": [
        { "has-other-names": "Yes" }
      ],
      "elements": ["cmpOtherNames"]
    }
  },
  {
    "controlElements": ["marital-status"],
    "requiredWhen": {
      "or": [
        { "marital-status": "Married" },
        { "marital-status": "Legally Separated" }
      ],
      "elements": [
        "spouse-given-name",
        "spouse-family-name",
        "date-of-marriage"
      ]
    }
  }
]
```

The flow control schema supports:

- Simple conditions: Show field B when field A equals 'Yes'
- OR conditions: Show fields when any of multiple conditions are met
- AND conditions: Show fields only when all conditions are met
- Multi-element control: One trigger field can show/hide multiple dependent fields
- Part-level control: Entire form sections can be shown/hidden based on conditions

Intelligent Component Recognition

One of the system's most valuable capabilities is recognizing that not all form fields are simple text inputs. When the AI encounters patterns indicating specialized field types, it extracts them as composite components that map directly to rich UI components.

Supported Component Types

When the AI encounters a mailing address section with fields for street, city, state, and ZIP code, it recognizes this as a composite 'address' component. The resulting schema element uses the 'cmpMailingAddress' ID, signaling to the UI that this should be rendered using a rich AddressField component that handles country-specific formatting, state/province selection, and ZIP code validation automatically.

Component Type Registry

```
// Simple Components (kebab-case IDs)
"text"              // Single-line text input
"textarea"          // Multi-line text input
"datepicker"        // Date selection with MM/DD/YYYY validation
"radiobuttongroup"  // Yes/No or multiple choice
"checkboxgroup"     // Multiple selection
"select"            // Dropdown selection
"countryselect"     // Country dropdown with ISO codes
"stateselect"       // US State/Territory dropdown

// Specialized Components (automatic pattern recognition)
"anumber"           // A-Number: A##########
"ssn"               // Social Security: ###-##-####
"uscisaccountnumber"  // 12-digit USCIS account
"phonenumber"       // Phone: (###) ###-####
// Composite Components (cmp prefix + camelCase IDs)
"fullname"          // Given, Middle, Last name fields
"address"           // Street, City, State, ZIP, Country
"applicantcertification"  // Signature and certification
"interpretercertification" // Language and signature
"preparercertification"   // Preparer information
```

Prompt-Driven Generation

The entire extraction process is controlled by prompts—detailed instructions that tell the AI exactly what to look for and how to structure the output. This approach offers significant advantages.

No Coding Required

The prompts are written in natural language with structured examples. A subject matter expert who understands the forms can maintain and update the prompts without programming knowledge.

Prompt Naming Convention Instructions

```
FIELD ID NAMING CONVENTIONS:

1. SIMPLE FIELDS: Use kebab-case
   Examples: "email-address", "date-of-birth", "phone-number"

2. COMPOSITE COMPONENTS: Use camelCase with 'cmp' prefix
   Full Name: "cmpCurrentLegalName", "cmpOtherNames"
   Address: "cmpMailingAddress", "cmpPhysicalAddress"

3. NESTED FIELDS (inside composites): Use camelCase
   Name fields: "givenName", "middleName", "lastName"
   Address fields: "addressLine1", "city", "state", "zipCode"
```

Version-Specific Prompts

Different form versions may have different

> This appoach maintains 95% accuracy even on the most complex government forms, where single-pass extraction would degrade significantly.

structures. Organizations can maintain a library of prompts:

- Base prompts for general extraction rules
- Form-specific prompts for unique structures (I-485, I-130, I-140, etc.)
- Version-specific prompts when form editions change
- Organization-specific prompts for custom validation requirements

Multi-Pass Extraction for Complex Forms

Large government forms present a challenge for any AI system: context limits. When a form spans 20+ pages with hundreds of fields, even advanced LLMs can lose track of details.

Our system addresses this through intelligent multi-pass extraction:

The multi-pass approach:

Multi-Pass Strategy Configuration

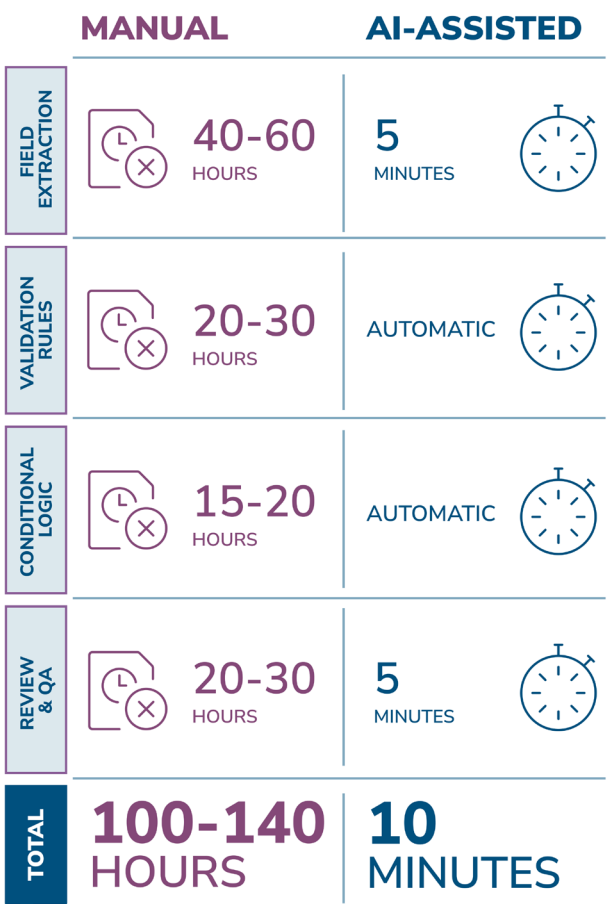
```
// PDF Analysis determines extraction strategy
{
  "pageCount": 24,
  "parts": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14],
  "strategy": {
    "type": "multi-pass",
    "passes": [
      { "parts": [1, 2, 3, 4, 5], "start": 1, "end": 5 },
      { "parts": [6, 7, 8, 9, 10], "start": 6, "end": 10 },
      { "parts": [11, 12, 13, 14], "start": 11, "end": 14 }
    ]
  }
}
```

- Automatically detects when multi-pass is needed (>10 parts or >20 pages)
- Groups parts into batches of approximately 5 for optimal accuracy
- Processes each batch independently with full AI attention
- Merges results while removing duplicates and sorting by part number
- Produces a single, coherent output as if extracted in one pass

This approach maintains 95%+ accuracy even on the most complex government forms, where single-pass extraction would degrade significantly.

Performance and Results

Compare this to traditional manual development:



Framework-Agnostic Integration

The three output schemas are designed to be consumed by any front-end framework. The data is pure JSON with no framework-specific dependencies.

React Integration Pattern

```
// React integration example
function DynamicForm({ formDef, validationSchema, flowControl }) {
  // Initialize validation with AJV
  const validate = ajv.compile(validationSchema);

  // Initialize flow control state
  const [visibleFields, setVisibleFields] = useState(
    new Set(formDef.elements.map(e => e.id))
  );

  // Render form elements dynamically
  return formDef.elements
    .filter(element => visibleFields.has(element.id))
    .map(element => (
      <FormField
        key={element.id}
        component={element.component}  // Maps to UI component
        id={element.id}
        label={element.label}
        required={element.required}
        onChange={(value) => {
          // Update flow control visibility
          updateVisibility(element.id, value, flowControl);
        }}
      />
    ));
}
```

The component mapping is straightforward:

- ‘text’ maps to TextInput component
- ‘fullname’ (cmpXxx) maps to FullNameField composite component
- ‘address’ (cmpXxx) maps to AddressField composite component
- ‘radiobuttongroup’ maps to RadioGroup component
- ‘datepicker’ maps to DatePicker with format validation

Because the component type is explicitly defined in the schema, the UI can map each element to the appropriate component without additional logic.

Conclusion

AI-assisted form digitization represents a fundamental shift in how organizations can approach form automation. This shift was only possible because of the foundation that came before it—years of disciplined engineering that defined exactly what the output needed to look like.

Our original deterministic approach proved the concept: PDF forms could be systematically converted into structured schemas that any UI framework could consume. The AI didn’t change what we were building, rather it changed how fast we could build it.

By combining the semantic understanding capabilities of LLMs with deterministic post-processing and schema generation, ReGenX achieves results that are both faster and more reliable than traditional manual approaches. The AI handles the extraction; our proven pipeline handles the transformation, validation, and schema generation.

I-485 FORM (24 pages, 14 parts)

Total Processing Time: **5-6 MINUTES**

Elements Generated: **200+ FIELDS**

Validation Rules: **AUTOMATICALLY GENERATED**

Multi-Pass Extraction **3 PASSES**

Conditional Logic Rules: **30+ FLOW CONTROL RULES**

Manual Review Required **~5 MINUTES FOR EDGE CASES**

The key innovations in this architecture include:

- Intelligent component recognition that understands field semantics, not just structure
- Automatic validation schema generation with conditional logic support
- Flow control extraction that captures the form's business rules
- Multi-pass extraction that maintains accuracy on complex documents
- Framework-agnostic output that integrates with any technology stack
- Prompt-driven approach that enables non-technical maintenance

The Bottom Line

What previously required weeks of skilled developer time can now be accomplished in minutes. Organizations dealing with dozens or hundreds of form types can dramatically reduce their digitization costs while improving consistency and maintainability.

The technology is production-ready today, processing complex government forms with 95%+ accuracy and generating complete, validated schemas that any front-end framework can consume directly.