# **Phase 1 Script → Agent MVP Evolution**

**Owner**: Assaf  
 **Starting Point**: Your working Phase 1 script (PDF → IR → Rules)  
 **Goal**: Gradually evolve into agent-based system, proving each step  
 **Timeline**: Small incremental changes, validate each step works

## **Current State: Phase 1 Script MVP**

You have a working script that:

* ✅ Takes PDF intelligence reports
* ✅ Extracts structured data with zero inference
* ✅ Generates rules in multiple formats
* ✅ Has compliance validation working

**Next**: Evolve this step-by-step into agent architecture

## **Evolution Step 1: Add Tool Abstraction (Week 1)**

**Hypothesis**: "I can wrap my existing functions as 'tools' without breaking anything"

### **Monday-Tuesday: Tool Wrapper Creation**

**Goal**: Convert existing functions into tool interface without changing behavior

# tools/pdf\_tool.py

class PDFTool:

def \_\_init\_\_(self):

# Your existing PDF processing code

pass

def execute(self, pdf\_path: str):

# Wrap your existing pdf processing function

return self.existing\_pdf\_function(pdf\_path)

# tools/extraction\_tool.py

class ExtractionTool:

def execute(self, text\_data, spans):

# Wrap your existing extraction logic

return self.existing\_extraction\_function(text\_data, spans)

#### **Tasks:**

* [ ] **Day 1**: Wrap PDF processing as PDFTool
* [ ] **Day 2**: Wrap IR extraction as ExtractionTool

### **Wednesday-Thursday: Test Tool Wrappers**

* [ ] **Day 3**: Run same FBI report through new tool interface
* [ ] **Day 4**: Verify outputs are identical to original script

### **Friday: Basic Tool Orchestration**

* [ ] **Day 5**: Create simple orchestrator that calls tools in sequence
* [ ] **Evening**: Validate entire pipeline produces same results

**Success Criteria**:

* [ ] Same input → Same output as Phase 1 script
* [ ] Code is now "tool-based" but functionally identical

## **Evolution Step 2: Add Agent Decision Making (Week 2)**

**Hypothesis**: "An agent can make simple decisions about which tools to call"

### **Monday-Tuesday: Simple Agent Logic**

**Goal**: Add basic decision-making without changing core functionality

# agents/simple\_agent.py

class SimpleAgent:

def \_\_init\_\_(self, tools):

self.tools = tools

def process(self, input\_data):

# Simple hardcoded decision tree

if input\_data.endswith('.pdf'):

pdf\_result = self.tools['pdf'].execute(input\_data)

extraction\_result = self.tools['extraction'].execute(pdf\_result)

return extraction\_result

#### **Tasks:**

* [ ] **Day 1**: Build SimpleAgent with hardcoded workflow
* [ ] **Day 2**: Test agent produces same results as tool orchestrator

### **Wednesday-Thursday: Add Basic Branching**

**Goal**: Agent makes simple decisions based on input type

def process(self, input\_data):

# Simple branching logic

if self.is\_pdf(input\_data):

return self.pdf\_workflow(input\_data)

elif self.is\_text(input\_data):

return self.text\_workflow(input\_data)

#### **Tasks:**

* [ ] **Day 3**: Add support for text input (bypass PDF processing)
* [ ] **Day 4**: Test with both PDF and text inputs

### **Friday: Validate Agent Behavior**

* [ ] **Day 5**: Run through multiple test cases, verify behavior matches expectations

**Success Criteria**:

* [ ] Agent makes correct tool choices based on input type
* [ ] Still produces same quality outputs as Phase 1 script

## **Evolution Step 3: Add Claude Integration (Week 3)**

**Hypothesis**: "I can use Claude for one specific task without breaking compliance"

### **Monday-Tuesday: Claude Driver Setup**

**Goal**: Add Claude capability but don't use it for critical extraction yet

# drivers/claude\_driver.py (from your existing compliance learnings)

class ClaudeDriver:

def \_\_init\_\_(self):

# Your proven compliance-first approach

pass

def safe\_completion(self, prompt, validation\_fn):

response = self.claude\_api\_call(prompt)

if not validation\_fn(response):

raise ComplianceError("Output failed validation")

return response

#### **Tasks:**

* [ ] **Day 1**: Set up Claude driver with your compliance validation
* [ ] **Day 2**: Test Claude on non-critical task (like report summarization)

### **Wednesday-Thursday: Integrate Claude for Enhancement**

**Goal**: Use Claude to enhance existing outputs, not replace core extraction

# Example: Use Claude to improve rule descriptions, not extract data

def enhance\_rules(self, existing\_rules):

# Use Claude to add better comments/documentation to rules

# Core rule logic still comes from your proven extraction

#### **Tasks:**

* [ ] **Day 3**: Use Claude to enhance rule documentation/comments
* [ ] **Day 4**: Test that Claude enhancement doesn't change rule logic

### **Friday: Validate Claude Integration**

* [ ] **Day 5**: Ensure Claude adds value without compromising compliance

**Success Criteria**:

* [ ] Claude improves output quality (better docs, clearer explanations)
* [ ] Zero compliance violations (core extraction still uses your proven method)

## **Evolution Step 4: Agent-Driven Workflow (Week 4)**

**Hypothesis**: "Agent can intelligently sequence tools based on document analysis"

### **Monday-Tuesday: Document Analysis Agent**

**Goal**: Agent analyzes document and decides processing strategy

class DocumentAnalysisAgent:

def analyze\_document(self, doc):

# Simple analysis: threat type, complexity, format

return {

'threat\_type': self.detect\_threat\_type(doc),

'complexity': self.assess\_complexity(doc),

'processing\_strategy': self.recommend\_strategy(doc)

}

#### **Tasks:**

* [ ] **Day 1**: Build document analysis capabilities
* [ ] **Day 2**: Test analysis accuracy on your golden dataset

### **Wednesday-Thursday: Dynamic Workflow Selection**

**Goal**: Agent chooses tools/sequence based on document analysis

def process\_document(self, doc):

analysis = self.analyze\_document(doc)

if analysis['threat\_type'] == 'financial\_fraud':

return self.financial\_workflow(doc)

elif analysis['threat\_type'] == 'romance\_scam':

return self.romance\_scam\_workflow(doc)

# etc.

#### **Tasks:**

* [ ] **Day 3**: Implement workflow selection logic
* [ ] **Day 4**: Test different document types trigger correct workflows

### **Friday: Validate Dynamic Processing**

* [ ] **Day 5**: Test that agent chooses appropriate processing for different threats

**Success Criteria**:

* [ ] Agent correctly identifies document types
* [ ] Different workflows produce better specialized outputs
* [ ] Overall quality meets or exceeds Phase 1 script

## **Evolution Step 5: Multi-Agent Collaboration (Week 5)**

**Hypothesis**: "Specialized agents working together produce better results"

### **Monday-Tuesday: Specialized Agents**

**Goal**: Break processing into specialized agents

# agents/extraction\_agent.py - Focuses only on data extraction

# agents/validation\_agent.py - Focuses only on compliance checking

# agents/compilation\_agent.py - Focuses only on rule generation

#### **Tasks:**

* [ ] **Day 1**: Create specialized extraction agent
* [ ] **Day 2**: Create specialized validation agent

### **Wednesday-Thursday: Agent Coordination**

**Goal**: Master agent coordinates specialized agents

class MasterAgent:

def process(self, doc):

# Coordinate specialized agents

extraction = self.extraction\_agent.extract(doc)

validation = self.validation\_agent.validate(extraction, doc)

if validation.passed:

return self.compilation\_agent.compile(extraction)

#### **Tasks:**

* [ ] **Day 3**: Build master coordination logic
* [ ] **Day 4**: Test multi-agent collaboration

### **Friday: Validate Multi-Agent System**

* [ ] **Day 5**: Compare multi-agent vs single-agent performance

**Success Criteria**:

* [ ] Multi-agent system produces higher quality outputs
* [ ] Specialization improves performance in each area
* [ ] System remains reliable and compliant

## **Evolution Step 6: Self-Improvement Loop (Week 6)**

**Hypothesis**: "Agent can learn from its mistakes and improve over time"

### **Monday-Tuesday: Feedback Collection**

**Goal**: Agent tracks its own performance and identifies improvement areas

class PerformanceTracker:

def track\_processing(self, input\_doc, output, user\_feedback):

# Track what worked and what didn't

self.store\_performance\_data(input\_doc, output, user\_feedback)

def identify\_improvement\_areas(self):

# Analyze patterns in errors/feedback

return self.analyze\_performance\_trends()

#### **Tasks:**

* [ ] **Day 1**: Build performance tracking system
* [ ] **Day 2**: Add user feedback collection points

### **Wednesday-Thursday: Self-Adjustment**

**Goal**: Agent adjusts its behavior based on tracked performance

class AdaptiveAgent:

def adjust\_processing\_strategy(self):

improvement\_areas = self.performance\_tracker.identify\_improvement\_areas()

for area in improvement\_areas:

self.optimize\_workflow(area)

#### **Tasks:**

* [ ] **Day 3**: Implement basic self-adjustment logic
* [ ] **Day 4**: Test agent adaptation on historical feedback

### **Friday: Validate Learning Loop**

* [ ] **Day 5**: Verify agent improves performance over time

**Success Criteria**:

* [ ] Agent identifies its own weak points
* [ ] Performance measurably improves with usage
* [ ] Self-improvement doesn't compromise compliance

## **Validation Checkpoints**

### **After Each Evolution Step:**

1. **Functional Test**: Same inputs produce same or better outputs
2. **Compliance Test**: Zero hallucinations, perfect source citations
3. **Performance Test**: Processing time doesn't degrade significantly
4. **Customer Test**: Show to 1-2 customers, get feedback

### **Final Validation (End of Week 6):**

* [ ] **Agent system outperforms original Phase 1 script**
* [ ] **Customers prefer agent version over script version**
* [ ] **System is ready for customer pilots**
* [ ] **Clear path to adding new capabilities through new agents/tools**

## **Key Principles for Each Step**

1. **Never break what works**: Each step must maintain existing functionality
2. **Prove before proceeding**: Don't move to next step until current step validates
3. **Keep compliance first**: Never compromise zero-hallucination guarantee
4. **Customer feedback**: Show progress to customers, get real feedback
5. **Measure improvement**: Each step should measurably improve some aspect

## **Success Metrics by Week**

**Week 1**: Tool abstraction complete, same outputs  
 **Week 2**: Agent decision-making working, same quality  
 **Week 3**: Claude integration adds value, zero compliance violations  
 **Week 4**: Dynamic workflows improve specialized threat processing  
 **Week 5**: Multi-agent collaboration improves overall quality  
 **Week 6**: Self-improvement loop demonstrates learning capability

**Final Goal**: Agent-based system that's demonstrably better than your Phase 1 script and ready for customer pilots.