Implementation Plan: OpenAI Function Calling for DecisionAgent

Executive Summary

This document outlines the plan to enhance the DecisionAgent with OpenAI Function Calling capabilities, transforming it from a fixed sequential pipeline into a dynamic, LLM-driven decision loop. This enhancement demonstrates modern AI agent patterns while maintaining the auditability and reliability required for RegTech compliance.

Table of Contents

---

1. Current State Analysis

1.1 Existing Architecture

The current AgentCheck system uses a **handcrafted sequential pipeline**:

```
┌─────────────────────────────────────────────────────────────────────┐
│                          ORCHESTRATOR                                │
│                                                                      │
│   Phase 1                Phase 2                Phase 3              │
│   ExtractionAgent  →     EmailAgent       →    DecisionAgent        │
│                                                                      │
│   [Fixed Order]          [Fixed Order]          [Fixed Order]        │
└─────────────────────────────────────────────────────────────────────┘
```

1.2 Current DecisionAgent Flow

```python
def run(self, incoming_email, extracted_fields, contact_found):
    # Step 1: ALWAYS analyze reply
    reply_analysis = self.tools.analyze_reply(reply, extracted_fields)
```

```
    # Step 2: ALWAYS decide compliance
    compliance_result, explanation =
self.tools.decide_compliance(reply_analysis)

    return result
```

Key Characteristics:

- ☑ Deterministic and predictable
- ☑ Easy to audit
- ❌ Cannot adapt to edge cases
- ❌ No self-correction capability
- ❌ Cannot escalate to human when needed
- ❌ LLM is only used as a "worker", not a "thinker"

1.3 Tool Execution Pattern

Aspect	Current Implementation
Who decides which tool?	Hard-coded in Python
Execution order	Fixed: analyze → decide
Number of iterations	Always exactly 2
Can skip tools?	No
Can retry tools?	No
Can call additional tools?	No

---

2. Problem Statement

2.1 Limitations of Fixed Pipeline

The current DecisionAgent cannot handle these real-world scenarios:

Scenario 1: Ambiguous Reply

```
University Reply: "Please provide the student ID number for verification."

Current Behavior:
  → analyze_reply() → "Unclear response"
  → decide_compliance() → INCONCLUSIVE

Desired Behavior:
  → analyze_reply() → "They need more info"
  → request_clarification() → Flag for follow-up
  → decide_compliance() → INCONCLUSIVE with actionable reason
```

```

```

Scenario 2: Suspicious Reply

```
Reply from: random.person@gmail.com (not @harvard.edu)

Current Behavior:
  → analyze_reply() → Process normally
  → decide_compliance() → May give false COMPLIANT

Desired Behavior:
  → analyze_reply() → "Sender domain suspicious"
  → escalate_to_human() → Flag for security review
  → DO NOT auto-decide (potential fraud)
```

Scenario 3: Clear Verified Response

```
University Reply: "We confirm this certificate is authentic."

Current Behavior:
  → analyze_reply() → "Verified"
  → decide_compliance() → COMPLIANT
  (2 LLM calls)

Desired Behavior:
  → decide_compliance() → COMPLIANT
  (1 LLM call - more efficient)
```

2.2 Gap Analysis

Requirement	Current	Needed
Handle ambiguity	❌	✅
Escalate to human	❌	✅
Adaptive tool selection	❌	✅
Self-correction	❌	✅
Efficiency (skip unnecessary steps)	❌	✅

---

3. Proposed Solution

3.1 Enhanced DecisionAgent with Function Calling

Transform DecisionAgent to use OpenAI Function Calling, enabling the LLM to **decide which tools to call** based on context.

```
┌─────────────────────────────────────────────────────────────────┐
│                    ENHANCED DECISION AGENT                      │
│                                                                 │
│   ┌─────────────────────────────────────────────────────────┐   │
│   │                         LLM                             │   │
│   │                                                         │   │
│   │   "I have these tools available. Based on the context,  │   │
│   │    I will decide which to call and in what order."      │   │
│   │                                                         │   │
│   │   Available Tools:                                      │   │
│   │   ├── analyze_reply()                                   │   │
│   │   ├── request_clarification()                           │   │
│   │   ├── escalate_to_human()                               │   │
│   │   └── decide_compliance()                               │   │
│   └─────────────────────────────────────────────────────────┘   │
│                            │                                    │
│                            ▼                                    │
│   ┌─────────────────────────────────────────────────────────┐   │
│   │                FUNCTION CALLING LOOP                    │   │
│   │                                                         │   │
│   │   while not done (max 5 iterations):                    │   │
│   │       1. LLM chooses a tool                             │   │
│   │       2. Execute the tool                               │   │
│   │       3. Return result to LLM                           │   │
│   │       4. LLM decides next action                        │   │
│   │                                                         │   │
│   │   Loop ends when:                                       │   │
│   │       - LLM calls decide_compliance(), OR               │   │
│   │       - LLM calls escalate_to_human(), OR               │   │
│   │       - Max iterations reached                          │   │
│   └─────────────────────────────────────────────────────────┘   │
└─────────────────────────────────────────────────────────────────┘
```

3.2 New Tool Definitions

```
DECISION_AGENT_TOOLS = [
    {
        "type": "function",
        "function": {
            "name": "analyze_reply",
            "description": "Analyze the university email reply to extract
verification status, tone, and key information. Use this when you need to
understand what the university is communicating.",
            "parameters": {
                "type": "object",
                "properties": {
```

```
                    "focus_areas": {
                        "type": "array",
                        "items": {"type": "string"},
                        "description": "Specific aspects to analyze: verification_status, sender_legitimacy, completeness, tone"
                    }
                },
                "required": []
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "request_clarification",
            "description": "Flag that the university reply is unclear or incomplete and additional information is needed. Use this when the reply doesn't provide enough information for a decision.",
            "parameters": {
                "type": "object",
                "properties": {
                    "reason": {
                        "type": "string",
                        "description": "Why clarification is needed"
                    },
                    "missing_information": {
                        "type": "array",
                        "items": {"type": "string"},
                        "description": "What specific information is missing"
                    },
                    "suggested_follow_up": {
                        "type": "string",
                        "description": "Recommended next action"
                    }
                },
                "required": ["reason"]
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "escalate_to_human",
            "description": "Escalate the case to a human compliance officer. Use this when: (1) potential fraud is detected, (2) the case is too complex for automated decision, (3) the reply is suspicious, or (4) high-stakes decision requires human oversight.",
            "parameters": {
                "type": "object",
                "properties": {
                    "reason": {
                        "type": "string",
                        "description": "Why human review is required"
```

```
                    },
                    "priority": {
                        "type": "string",
                        "enum": ["LOW", "MEDIUM", "HIGH", "CRITICAL"],
                        "description": "Urgency level"
                    },
                    "risk_indicators": {
                        "type": "array",
                        "items": {"type": "string"},
                        "description": "Specific concerns identified"
                    }
                },
                "required": ["reason", "priority"]
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "decide_compliance",
            "description": "Make the final compliance decision. Only call this when you have sufficient information to make a confident decision.",
            "parameters": {
                "type": "object",
                "properties": {
                    "status": {
                        "type": "string",
                        "enum": ["COMPLIANT", "NOT_COMPLIANT", "INCONCLUSIVE"],
                        "description": "Final compliance status"
                    },
                    "confidence_score": {
                        "type": "number",
                        "minimum": 0,
                        "maximum": 1,
                        "description": "Confidence in the decision (0.0 to 1.0)"
                    },
                    "explanation": {
                        "type": "string",
                        "description": "Detailed reasoning for the decision"
                    },
                    "evidence_summary": {
                        "type": "string",
                        "description": "Key evidence supporting the decision"
                    }
                },
                "required": ["status", "confidence_score", "explanation"]
            }
        }
    }
]
```

3.3 Example Flows

Flow 1: Clear Verification (Efficient Path)

```
Input: "We hereby confirm that John Doe graduated with a Bachelor of Science
        degree on May 15, 2023. Certificate number: CS-2023-1234."

LLM thinks: "This is a clear confirmation with specific details.
             I can make a decision directly."

Iteration 1:
  LLM calls: decide_compliance(
      status="COMPLIANT",
      confidence_score=0.95,
      explanation="University explicitly confirmed graduation with matching
details"
  )

Result: COMPLIANT (1 tool call, efficient!)
```

Flow 2: Ambiguous Reply (Adaptive Path)

```
Input: "Please provide additional documentation including the student ID
        and certificate serial number for our verification process."

LLM thinks: "This isn't a yes or no - they need more information."

Iteration 1:
  LLM calls: analyze_reply(focus_areas=["verification_status", "completeness"])
  Result: {status: "pending", requires_additional_info: true}

Iteration 2:
  LLM calls: request_clarification(
      reason="University requires additional documentation",
      missing_information=["student_id", "certificate_serial_number"],
      suggested_follow_up="Collect missing documents and resubmit"
  )

Iteration 3:
  LLM calls: decide_compliance(
      status="INCONCLUSIVE",
      confidence_score=0.85,
      explanation="Verification pending - university requires additional
documentation"
  )

Result: INCONCLUSIVE with actionable follow-up (3 tool calls)
```

Flow 3: Suspicious Activity (Safety Path)

```
Input: Reply from suspicious@gmail.com (expected: verification@harvard.edu)
       "Yes the certificate is valid."

LLM thinks: "The sender domain doesn't match the university. This could be
fraud."

Iteration 1:
  LLM calls: analyze_reply(focus_areas=["sender_legitimacy",
"verification_status"])
  Result: {sender_suspicious: true, domain_mismatch: true}

Iteration 2:
  LLM calls: escalate_to_human(
      reason="Sender email domain does not match university domain - potential
fraud",
      priority="HIGH",
      risk_indicators=["domain_mismatch", "external_sender",
"overly_brief_response"]
  )

Result: ESCALATED - No automated decision made (2 tool calls, safety preserved)
```

---

4. Why Function Calling Over LangChain

4.1 The Decision Context

The assignment states:

> "You may use: LangChain, LlamaIndex, OpenAI function calling, Your own handcrafted agent loop"
> "We evaluate your engineering thinking, not which framework you pick."

This suggests the interviewer wants to see **thoughtful technology selection**, not just framework adoption.

4.2 Comparison Matrix

Aspect	LangChain	OpenAI Function Calling
Dependencies	Heavy (~50+ packages)	Minimal (just openai)
Learning curve	Steep	Gentle
Abstraction level	High (can hide important details)	Low (full control)
Debugging	Complex (many layers)	Simple (direct API)
Vendor lock-in	LangChain-specific patterns	OpenAI API standard

Aspect	LangChain	OpenAI Function Calling
Customization	Framework constraints	Full flexibility
Overhead	Significant	Minimal
Production readiness	Requires careful config	Direct to production

4.3 Why Function Calling is Better for This Project

Reason 1: Right-Sized Solution

```
Project Scope:
├── 3 agents
├── ~10 tools
├── 1 workflow
└── Prototype/Demo

LangChain is designed for:
├── Complex multi-chain workflows
├── Dozens of integrations
├── Memory management
├── Vector stores
└── Enterprise scale

→ LangChain is OVERKILL for this scope
```

Reason 2: Demonstrates Core Understanding

```
Using LangChain:
  → "I used AgentExecutor with tools" (black box)

Using Function Calling directly:
  → "I implemented a ReAct-style loop with explicit tool routing"
  → "Here's how I handle the conversation history"
  → "Here's my termination condition logic"
  → Shows UNDERSTANDING, not just USAGE
```

Reason 3: Compliance & Auditability

```
LangChain:
  └── Many abstraction layers
      └── Callbacks spread across modules
          └── Harder to ensure complete audit trail

Function Calling:
```

```
    └── Single loop, single file
        └── Every decision point visible
            └── Complete audit trail guaranteed
```

Reason 4: Maintainability

```python
# LangChain approach
from langchain.agents import AgentExecutor, create_openai_functions_agent
from langchain.tools import Tool
from langchain.memory import ConversationBufferMemory
from langchain.prompts import ChatPromptTemplate
from langchain.callbacks import BaseCallbackHandler
# ... 10+ more imports

agent = AgentExecutor(
    agent=create_openai_functions_agent(llm, tools, prompt),
    tools=tools,
    memory=memory,
    callbacks=[handler],
    verbose=True,
    max_iterations=5,
    early_stopping_method="generate"
)
# What happens inside? 🤔

# Function Calling approach (what we'll implement)
while iterations < max_iterations:
    response = llm.chat(messages, tools=tools)
    if response.tool_calls:
        result = execute_tool(response.tool_calls[0])
        messages.append(tool_result)
    else:
        break
# Crystal clear what's happening ✓
```

Reason 5: Future Flexibility

```
Function Calling implementation can be:
├── Migrated to LangChain later if needed
├── Adapted to other providers (Anthropic, Google)
├── Extended with custom logic easily
└── Tested at every step

LangChain implementation:
├── Locked into LangChain patterns
├── Major refactor to change approach
└── Framework updates may break code
```

4.4 Summary: Engineering Justification

Interview Talking Point	Function Calling	LangChain
"Why this choice?"	"Right-sized for scope, shows understanding"	"It's popular"
"How does it work?"	Can explain every line	"The framework handles it"
"Can you modify X?"	"Yes, it's just a loop"	"Need to check framework docs"
"What about scale?"	"Can add LangChain later if needed"	"Already using it"
"Audit compliance?"	"Full control over logging"	"Need to configure callbacks"

5. Implementation Plan

5.1 Overview

```
┌─────────────────────────────────────────────────────────────────────┐
│                    IMPLEMENTATION PHASES                             │
│                                                                      │
│   Phase 1           Phase 2           Phase 3           Phase 4     │
│   ─────────         ─────────         ─────────         ─────────   │
│   Preparation       Core Loop         Integration       Testing     │
│   (0.5 day)         (1 day)           (0.5 day)         (0.5 day)   │
│                                                                      │
│   Total Estimated Time: 2-3 days                                    │
└─────────────────────────────────────────────────────────────────────┘
```

5.2 Phase 1: Preparation (0.5 day)

Task 1.1: Add Function Calling Support to LLMClient

File: `api/utils/llm_client.py`

```
def complete_with_tools(
    self,
    messages: List[Dict],
    tools: List[Dict],
    tool_choice: str = "auto"
) -> Dict[str, Any]:
    """
    Call LLM with function calling capability.
```

```
    Args:
        messages: Conversation history
        tools: Tool definitions in OpenAI format
        tool_choice: "auto", "none", or specific tool

    Returns:
        Response with potential tool_calls
    """
    response = self.client.chat.completions.create(
        model=self.model,
        messages=messages,
        tools=tools,
        tool_choice=tool_choice
    )
    return response.choices[0].message
```

Task 1.2: Create Tool Definitions

File: `api/tools/decision_tools.py` (new file)

Define the 4 tools for DecisionAgent:

- `analyze_reply`
- `request_clarification`
- `escalate_to_human`
- `decide_compliance`

5.3 Phase 2: Core Implementation (1 day)

Task 2.1: Create Function Calling Loop

File: `api/agents/decision_agent_fc.py` (new file)

```
class DecisionAgentWithFunctionCalling:
    """
    Enhanced DecisionAgent using OpenAI Function Calling.

    Unlike the original DecisionAgent which follows a fixed pipeline,
    this agent uses LLM to dynamically decide which tools to call
    based on the context.
    """

    def run(self, incoming_email, extracted_fields, max_iterations=5):
        messages = self._build_initial_messages(incoming_email,
extracted_fields)

        for i in range(max_iterations):
            # Log iteration start
            self.audit.log_step(
                step=f"fc_iteration_{i+1}",
```

```
                action="LLM deciding next action",
                agent=self.AGENT_NAME
            )

            # Call LLM with tools
            response = self.llm.complete_with_tools(
                messages=messages,
                tools=DECISION_TOOLS
            )

            if response.tool_calls:
                # Execute tool and continue loop
                tool_call = response.tool_calls[0]
                result = self._execute_tool(tool_call)
                messages = self._append_tool_result(messages, response, result)

                # Check for terminal tools
                if tool_call.function.name in ["decide_compliance",
"escalate_to_human"]:
                    break
            else:
                # LLM finished without tool call
                break

        return self._build_final_result(messages)
```

Task 2.2: Implement Tool Executors

```
def _execute_tool(self, tool_call) -> Dict:
    """Route tool call to appropriate handler."""
    name = tool_call.function.name
    args = json.loads(tool_call.function.arguments)

    handlers = {
        "analyze_reply": self._handle_analyze_reply,
        "request_clarification": self._handle_request_clarification,
        "escalate_to_human": self._handle_escalate_to_human,
        "decide_compliance": self._handle_decide_compliance
    }

    result = handlers[name](**args)

    # Log tool execution
    self.audit.log_step(
        step=f"tool_execution_{name}",
        action=f"Executed tool: {name}",
        tool=name,
        input_data=args,
        output_data=result
    )
```

```python
    return result
```

5.4 Phase 3: Integration (0.5 day)

Task 3.1: Update Orchestrator

File: `api/agents/orchestrator.py`

Add configuration to choose between original and function-calling agent:

```python
class AgentOrchestrator:
    def __init__(self, use_function_calling: bool = False, ...):
        if use_function_calling:
            self.decision_agent = DecisionAgentWithFunctionCalling(...)
        else:
            self.decision_agent = DecisionAgent(...)
```

Task 3.2: Update API Endpoint

File: `api/main.py`

```python
@app.post("/verify")
async def verify_certificate(
    request: VerificationRequest,
    use_function_calling: bool = Query(default=False, description="Use enhanced
agent")
):
    orchestrator = AgentOrchestrator(use_function_calling=use_function_calling)
    return orchestrator.verify_certificate(...)
```

Task 3.3: Update Schemas

File: `api/models/schemas.py`

Add new fields for escalation and clarification:

```python
class ComplianceReport(BaseModel):
    # ... existing fields
    escalated_to_human: bool = False
    escalation_reason: Optional[str] = None
    clarification_needed: bool = False
    missing_information: Optional[List[str]] = None
    function_calling_enabled: bool = False
    tool_calls_made: List[str] = []
```

5.5 Phase 4: Testing (0.5 day)

Task 4.1: Unit Tests

File: `tests/test_decision_agent_fc.py`

```
class TestDecisionAgentFunctionCalling:
    def test_clear_verified_single_call(self):
        """Clear verification should result in single decide_compliance call."""

    def test_ambiguous_reply_multiple_calls(self):
        """Ambiguous reply should trigger analysis before decision."""

    def test_suspicious_reply_escalation(self):
        """Suspicious reply should escalate to human."""

    def test_max_iterations_safety(self):
        """Should not exceed max iterations."""

    def test_audit_trail_complete(self):
        """All tool calls should be logged."""
```

Task 4.2: Integration Tests

Test end-to-end flow with function calling enabled.

---

6. Technical Specifications

6.1 New Files to Create

File	Purpose
`api/tools/decision_tools.py`	Tool definitions for function calling
`api/agents/decision_agent_fc.py`	New DecisionAgent with function calling
`tests/test_decision_agent_fc.py`	Unit tests

6.2 Files to Modify

File	Changes
`api/utils/llm_client.py`	Add `complete_with_tools()` method
`api/agents/orchestrator.py`	Add toggle for function calling agent
`api/models/schemas.py`	Add escalation/clarification fields

File	Changes
`api/main.py`	Add query parameter for agent type
`RESEARCH_INSIGHT.md`	Document the enhancement

6.3 API Changes

```
POST /verify
  Query Parameters:
    - use_function_calling: bool (default: false)

  Response additions:
    - escalated_to_human: bool
    - escalation_reason: string | null
    - clarification_needed: bool
    - missing_information: string[] | null
    - function_calling_enabled: bool
    - tool_calls_made: string[]
```

---

7. Testing Strategy

7.1 Test Scenarios

Scenario	Input	Expected Behavior
Clear Verified	Explicit confirmation email	1 tool call → COMPLIANT
Clear Rejected	Explicit rejection email	1-2 tool calls → NOT_COMPLIANT
Ambiguous	Request for more info	2-3 tool calls → INCONCLUSIVE
Suspicious Sender	Wrong email domain	2 tool calls → ESCALATED
Complex Case	Multiple concerns	3-4 tool calls → Appropriate result
No Reply	Empty/null email	1 tool call → INCONCLUSIVE

7.2 Audit Trail Verification

Each test should verify:

- All tool calls are logged
- Input/output data is recorded
- Timestamps are sequential
- Agent name is correct

---

8. Risks and Mitigations

Risk	Likelihood	Impact	Mitigation
LLM makes unexpected tool choices	Medium	Medium	Max iterations limit, terminal tool detection
Infinite loop	Low	High	Hard cap at 5 iterations
Higher API costs	Medium	Low	Efficient prompting, caching
Inconsistent behavior	Medium	Medium	Temperature=0, clear tool descriptions
Breaking existing tests	Low	Medium	Keep original agent, use feature flag

---

9. Success Criteria

9.1 Functional Requirements

- [ ] Function calling loop executes correctly
- [ ] All 4 tools are callable and produce expected results
- [ ] Escalation path works for suspicious cases
- [ ] Original agent still works (backward compatible)
- [ ] Audit trail captures all tool calls

9.2 Non-Functional Requirements

- [ ] Max 5 iterations enforced
- [ ] Average execution time < 10 seconds
- [ ] All tool executions logged
- [ ] Clear error messages for edge cases

9.3 Documentation Requirements

- [ ] RESEARCH_INSIGHT.md updated with reasoning
- [ ] Code comments explain function calling logic
- [ ] README updated with new feature flag

Conclusion

This implementation plan transforms the DecisionAgent from a fixed pipeline into a dynamic, LLM-driven agent while:

1. **Maintaining Simplicity** - Using OpenAI Function Calling directly instead of heavy frameworks
2. **Preserving Auditability** - Every tool call is logged with inputs/outputs
3. **Adding Flexibility** - LLM can adapt to edge cases
4. **Ensuring Safety** - Escalation path for suspicious cases
5. **Demonstrating Understanding** - Shows knowledge of AI agent patterns, not just framework usage

The choice of Function Calling over LangChain is a deliberate engineering decision that prioritizes:

- Right-sized solution for the project scope
- Full control and visibility
- Easier debugging and maintenance
- Clear demonstration of core concepts

Document Version: 1.0.0
Created: December 2024