Research & Engineering Insight Document

AgentCheck: AI Agent for Automated Qualification Verification

Table of Contents

1. Research Insight

1.1 AI Agent Patterns Studied

During the design phase, I researched several prominent AI agent architectures:

Pattern	Description	Pros	Cons
ReAct (Reasoning + Acting)	Agent alternates between reasoning and taking actions	Clear thought process, interpretable	Can be verbose, slower
Function Calling	LLM decides which tool to use via structured outputs	Reliable, deterministic tool selection	Limited reasoning visibility
Plan-and-Execute	Agent creates plan first, then executes steps	Good for complex tasks	Plan may become stale
Multi-Agent Systems	Specialized agents collaborate	Modularity, separation of concerns	Coordination overhead
Tool-Use Agents	Agent has access to tools and decides when to use them	Flexible, extensible	Requires good tool design

1.2 Why I Chose Multi-Agent + Function Calling

I selected a **Multi-Agent Architecture with Function Calling** for the following reasons:

1. **Separation of Concerns**: Each agent has a clear responsibility
    - Extraction Agent: Document processing expertise
    - Email Agent: Communication handling
    - Decision Agent: Compliance reasoning

2. **Auditability**: In RegTech, audit trails are critical. Having distinct agents makes it easy to track who did what and why.
3. **Reliability**: Function calling provides structured tool invocation, reducing hallucination risks compared to free-form agent reasoning.
4. **Modularity**: Agents can be improved, tested, or replaced independently.
5. **Compliance Alignment**: The workflow mirrors how a human compliance officer would work, making it easier to explain to auditors.

1.3 Alternatives Considered

Alternative 1: Single Monolithic Agent

```
PDF → Single Agent (does everything) → Report
```

Why Rejected:

- Hard to audit individual steps
- Single point of failure
- Difficult to maintain and test
- Violates single responsibility principle

Alternative 2: Pure ReAct Pattern

```
While not complete:
    Thought → Action → Observation → Thought → ...
```

Why Rejected:

- Less deterministic
- Harder to guarantee all required steps are executed
- More expensive (many LLM calls)
- Audit trail less structured

Alternative 3: Fully Autonomous Agent

Let the agent decide the entire workflow dynamically.

Why Rejected:

- Too unpredictable for compliance use cases
- Regulatory requirements need deterministic behavior
- Harder to explain decisions to auditors

1.4 Trade-offs Analysis

Aspect	Our Approach	Trade-off
Determinism	High - Fixed workflow steps	Less flexibility for edge cases
Auditability	Excellent - Every step logged	More storage/logging overhead
Tool Execution	Structured function calls	Need to maintain tool definitions
LLM Dependency	For extraction/analysis only	Graceful fallback when LLM unavailable
Agent Autonomy	Limited - Orchestrator controls flow	May miss optimization opportunities

2. Engineering Solution Thinking

2.1 Architecture Decisions

Decision 1: Tools as First-Class Citizens

Each tool is a discrete, testable unit:

```
def parse_pdf(pdf_path: str) -> Dict[str, Any]
def extract_fields(raw_text: str) -> ExtractedFields
def identify_university(extracted_fields: ExtractedFields) -> str
def lookup_contact(university_name: str) -> Optional[UniversityContact]
def draft_email(...) -> Dict[str, str]
def send_to_outbox(...) -> OutgoingEmail
def read_reply(...) -> IncomingEmail
def analyze_reply(...) -> ReplyAnalysis
def decide_compliance(...) -> Tuple[ComplianceResult, str]
```

Rationale:

- Tools can be unit tested independently
- Easy to add new tools without modifying agents
- Clear interface contracts
- Audit logger wraps each tool call automatically

Decision 2: Prompt Templates (Jinja2)

Prompts are externalized to `config/prompts/*.j2`:

```
# extract_fields.j2
Extract the following fields from the certificate text:
- candidate_name: {{ instructions }}
...
Certificate Text: {{ certificate_text }}
```

Rationale:

- Non-engineers can modify prompts
- Version control for prompt changes
- Easy A/B testing of prompts
- Separation of logic and content

Decision 3: Config-Driven University Mappings

```
{
  "universities": {
    "University of Example": {
      "email": "verification@example.edu",
      "country": "USA",
      "verification_department": "Office of the Registrar"
    }
  }
}
```

Rationale:

- Business users can add universities without code changes
- Easy to integrate with external data sources
- Supports different verification endpoints per university

2.2 Balancing Hard-Coded Logic vs. Generative Reasoning

Step	Approach	Reason
Workflow order	Hard-coded	Must be consistent for audit
Field extraction	LLM	Handles variations in certificate formats
University lookup	Database + fuzzy match	Deterministic, fast
Email drafting	LLM	Natural language generation
Reply analysis	LLM	Handles varied response formats
Compliance mapping	Hard-coded	Regulatory requirement

Key Insight: Use LLM where natural language understanding is needed; use deterministic code where consistency is required.

2.3 Failure Cases and Mitigations

Failure Case	Mitigation
PDF unreadable	Fallback to text file; return INCONCLUSIVE

Failure Case	Mitigation
LLM unavailable	Mock responses for demo; queue for retry
University not found	Mark as INCONCLUSIVE; log for manual review
Ambiguous reply	Lower confidence score; may need human review
LLM hallucination	Use structured output; validate response schema
Network timeout	Retry with exponential backoff
Rate limiting	Task queue with throttling

2.4 Audit Trail Design

Every action creates an `AuditLogEntry`:

```
@dataclass
class AuditLogEntry:
    timestamp: datetime
    step: str          # "001_parse_pdf"
    action: str        # Human-readable description
    agent: str         # Which agent performed this
    tool: str          # Which tool was used
    input_data: Dict   # Sanitized inputs
    output_data: Dict  # Sanitized outputs
    success: bool
    error_message: Optional[str]
```

Key Features:

- Sequential step numbering for timeline reconstruction
- Sensitive data automatically redacted
- Stored as JSONL for streaming writes
- Summary file generated at session end

---

3. Real-World Applicability

3.1 Extension to Real Email Integration

Current: Simulated outbox/inbox

Production Path:

```
class RealEmailService(EmailService):
    def send_email(self, email: OutgoingEmail) -> str:
        # Use SendGrid/AWS SES/SMTP
        response = sendgrid.send(
```

```
            to=email.recipient_email,
            subject=email.subject,
            body=email.body
        )
        return response.message_id

    def poll_inbox(self, reference_id: str) -> Optional[IncomingEmail]:
        # Use IMAP/Gmail API/Webhook
        messages = imap.search(subject=f"RE: {reference_id}")
        return self._parse_message(messages[0]) if messages else None
```

Considerations:

- Email sending rate limits
- Reply detection (subject matching, thread tracking)
- Spam folder handling
- Bounce handling

3.2 Real Certificate OCR

Current: Text extraction from PDF

Production Enhancement:

```
class OCREnhancedParser(PDFParser):
    def parse_pdf(self, pdf_path: str) -> Dict:
        # Try text extraction first
        text = self._extract_text(pdf_path)

        if len(text.strip()) < 50:  # Likely scanned image
            text = self._ocr_extract(pdf_path)

        return {"raw_text": text, "ocr_used": True}

    def _ocr_extract(self, pdf_path: str) -> str:
        # Use Tesseract/AWS Textract/Google Vision
        return textract.process(pdf_path)
```

3.3 API-Based University Verification

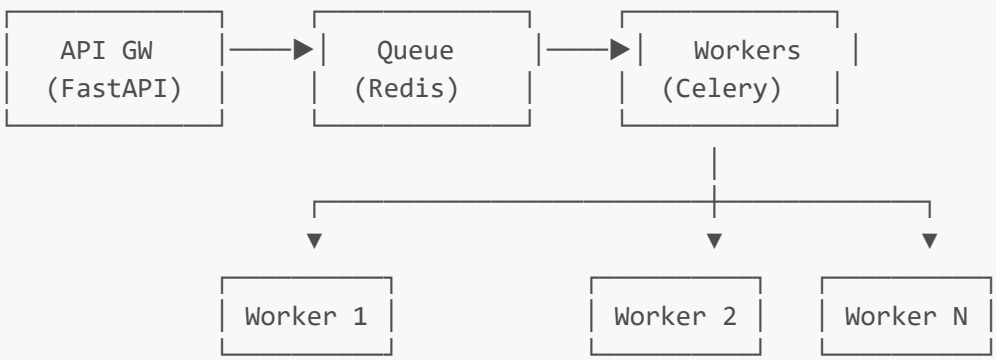
Many universities offer verification APIs:

```
class UniversityAPIVerifier:
    SUPPORTED_APIS = {
        "national_clearinghouse": NationalClearinghouseAPI,
        "hedd_uk": HEDDVerificationAPI,
        "direct_api": DirectUniversityAPI
    }
```

```
    async def verify(self, certificate: ExtractedFields) -> VerificationResult:
        api = self._get_api_for_university(certificate.university_name)
        return await api.verify(
            name=certificate.candidate_name,
            degree=certificate.degree_name,
            date=certificate.issue_date
        )
```

3.4 Scaling to 1,000+ Checks/Day

Architecture for Scale:



Key Components:

- **Redis Queue:** Decouple request from processing
- **Celery Workers:** Horizontal scaling
- **Rate Limiter:** Per-university rate limiting
- **Retry Logic:** Exponential backoff
- **Dead Letter Queue:** Failed tasks for manual review

3.5 Monitoring, SLA, and Retries

```
# Example monitoring setup
class VerificationMetrics:
    def __init__(self):
        self.processing_time = Histogram('verification_duration_seconds')
        self.result_counter = Counter('verification_results_total', ['result'])
        self.error_counter = Counter('verification_errors_total',
['error_type'])

    def track_verification(self, report: ComplianceReport):
        self.processing_time.observe(report.processing_time_seconds)
        self.result_counter.labels(result=report.compliance_result.value).inc()

# SLA Configuration
SLA_CONFIG = {
```

```
    "max_processing_time_seconds": 30,
    "max_retry_attempts": 3,
    "retry_backoff_seconds": [10, 30, 60],
    "alert_threshold_queue_size": 100
}
```

---

4. Security & Compliance Thoughts

4.1 Data Privacy

Personal Data Handled:

- Candidate names
- Degree information
- Certificate images

Protections Implemented:

- Audit logs sanitize sensitive data
- No PII in log messages
- Data stored locally (no third-party storage)

Production Recommendations:

- Encrypt data at rest (AES-256)
- Encrypt in transit (TLS 1.3)
- Implement data retention policies
- GDPR/CCPA compliance (right to deletion)
- PII masking in logs

4.2 Model Hallucination Risks

Identified Risks:

1. LLM fabricates certificate fields
2. LLM misinterprets university reply
3. LLM invents confidence scores

Mitigations:

```
# 1. Schema validation
def extract_fields(raw_text: str) -> ExtractedFields:
    response = llm.complete_json(prompt)
    # Validate against known patterns
    if not self._validate_date_format(response.get("issue_date")):
        response["issue_date"] = None
    return ExtractedFields(**response)
```

```
# 2. Confidence thresholds
if analysis.confidence_score < 0.7:
    result = VerificationStatus.INCONCLUSIVE
    # Flag for human review

# 3. Fallback to keyword analysis
def _fallback_analyze_reply(self, reply_text: str) -> ReplyAnalysis:
    # Deterministic keyword matching as backup
    ...
```

4.3 Traceability Requirements

Compliance Standard Alignment:

Requirement	Implementation
Who made the decision?	`agent` field in audit log
When?	`timestamp` field (UTC)
What inputs?	`input_data` field
What outputs?	`output_data` field
Why?	`decision_explanation` field
Can it be reproduced?	Session ID links all related logs

Report Retention:

- Reports stored as JSON files
- Session summaries for quick lookup
- JSONL logs for detailed reconstruction

4.4 Human-in-the-Loop Necessity

When Human Review is Required:

1. **Low Confidence Results**

```
if analysis.confidence_score < CONFIDENCE_THRESHOLD:
    report.requires_human_review = True
```

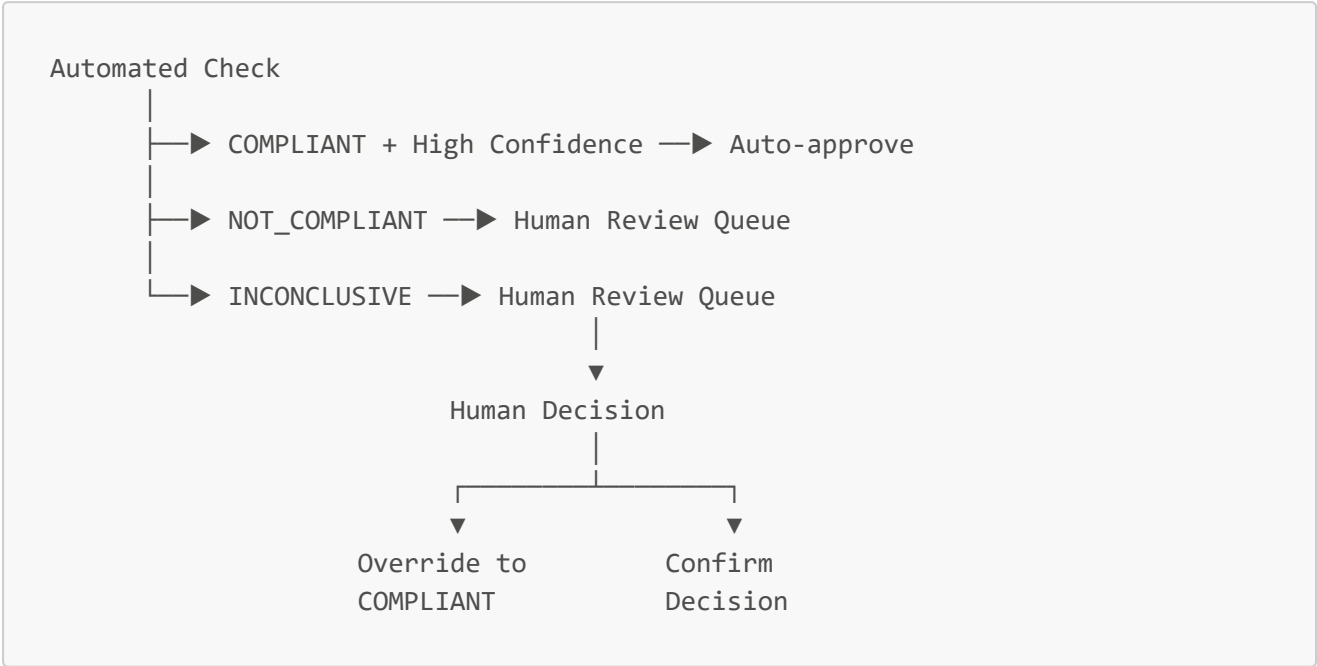
2. **INCONCLUSIVE Results**
    - University not in database
    - Ambiguous reply received
    - PDF parsing failed
3. **High-Stakes Decisions**

- NOT_COMPLIANT results affecting employment
- First-time university verification

4. **Anomaly Detection**

```
if self._detect_anomaly(certificate):
    # Unusual patterns trigger review
    report.flagged_for_review = True
```

Production Workflow:



Conclusion

This document outlines the research, engineering decisions, and practical considerations for the AgentCheck certificate verification system. The multi-agent architecture with function calling provides a balance of:

- **Reliability** through structured tool execution
- **Flexibility** through LLM-powered analysis
- **Auditability** through comprehensive logging
- **Extensibility** through modular design

For production deployment, the key enhancements would be:

1. Real email integration
2. University API connections
3. Enhanced OCR capabilities
4. Horizontal scaling with message queues
5. Robust human-in-the-loop workflows

The system is designed with RegTech compliance requirements at its core, ensuring that every decision can be traced, explained, and audited.

Document Version: 1.0.0
Last Updated: December 2024