N8N Workflow Upgrade Plan

Step-by-Step Guide for Construction AI Platform (v1.119.2)

Executive Summary

Current State: 45+ workflows across CAD/BIM and construction management
Target: Unified master agent with webhook + Telegram, no empty workflows, all v1.119.2 compatible
Timeline: 3-4 weeks
Complexity: Medium (straightforward but requires systematic testing)

Phase 1: Preparation (Days 1-2)

Step 1.1: Audit Existing Workflows

Time: 4 hours

Create a spreadsheet of all 45+ workflows:

```
Workflow Name | Current Status | Node Count | Issues | Compatibility | Priority
──────────────────────────────────────────────────────────────────────────────
01_Convert_DWG | Active | 12 | None | ✓ | HIGH
02_Data_Extract | Error | 8 | Undefined nodes | ✗ | HIGH
03_Classify_BIM | Active | 15 | None | ✓ | MEDIUM
... | ... | ... | ... | ... | ...
```

Tasks:

- [ ] Export all workflows from n8n (Settings → Export → Download)
- [ ] Open each workflow JSON and check for:
  - Undefined nodes (search for "type": null or missing node definitions)
  - Old node types (Function, Script, HTTPRequest v1)
  - Custom community nodes
  - Missing dependencies
- [ ] Document all issues in spreadsheet
- [ ] Categorize by criticality:
  - **CRITICAL:** Master agent, main data extraction, core CAD processing
  - **HIGH:** Secondary extraction, classification
  - **MEDIUM:** Utility, reporting, optional features

- **LOW:** Deprecated, rarely used

Output: spreadsheet_workflow_audit.xlsx

Step 1.2: Set Up Testing Environment

Time: 3 hours

- [ ] Create a separate n8n instance (staging) for testing (don't modify production)
- [ ] Or create a dedicated folder in existing n8n: "Workflows - Upgrade Testing"
- [ ] Backup all production workflows (export as JSON files)
- [ ] Create a test Telegram bot for testing (get API key from BotFather)
- [ ] Create test webhook endpoints (ngrok or similar if testing locally)

Validation:

- [ ] Staging n8n instance accessible
- [ ] All production workflows backed up locally
- [ ] Test webhook working
- [ ] Test Telegram bot responding

Step 1.3: Document Current Integration Points

Time: 2 hours

- [ ] Map which workflows call which other workflows
- [ ] List all external APIs used (OpenAI, Gemini, OCR.space, etc.)
- [ ] Document all webhook endpoints currently in use
- [ ] List all Telegram workflows (if any)
- [ ] Identify data flow between workflows

Output: integration_map.json

Phase 2: Node Compatibility Check & Fix (Days 3-5)

Step 2.1: Identify Problematic Nodes

Time: 8 hours

For each workflow, open in JSON editor and check:

Common Issues in v1.119.2:

1. **Old Function/Code Nodes**
   - Old: `"type": "n8n-nodes-base.function"`

- New: `"type": "n8n-nodes-base.code"`
- **Fix:** Use new Code node, test JavaScript syntax compatibility

2. **HTTP Request Node**
   - Old: `"type": "n8n-nodes-base.httpRequest"` (v1)
   - New: `"type": "n8n-nodes-base.httpRequest"` (v2, different parameters)
   - **Fix:** Check authentication, headers, parameters format
3. **Missing Nodes**
   - Community nodes not installed
   - **Fix:** Install via n8n node marketplace or npm
4. **Spreadsheet/Excel**
   - Old: `"type": "n8n-nodes-base.spreadsheet"`
   - New: Use `n8n-nodes-base.readBinaryFile` + `n8n-nodes-base.spreadsheetFile`
   - **Fix:** Split into separate nodes

Diagnostic Script:

```
import json
import os

def audit_workflow_nodes(workflow_json_path):
    with open(workflow_json_path, 'r') as f:
        workflow = json.load(f)

    issues = []
    nodes = workflow.get('nodes', [])

    for node in nodes:
        node_type = node.get('type', 'UNKNOWN')
        node_name = node.get('name', 'unnamed')

        # Check for undefined nodes
        if not node_type or node_type == 'null':
            issues.append(f"UNDEFINED: {node_name} - no type")

        # Check for deprecated types
        deprecated = [
            'n8n-nodes-base.function',
            'n8n-nodes-base.script',
            'n8n-nodes-base.httpRequest' # v1
        ]

        if any(dep in node_type for dep in deprecated):
            issues.append(f"DEPRECATED: {node_name} ({node_type})")

        # Check for null parameters
        params = node.get('parameters', {})
        if params is None:
            issues.append(f"NULL PARAMS: {node_name}")
```

```
    return issues

# Run audit on all workflows
workflow_dir = '/path/to/workflows'
for workflow_file in os.listdir(workflow_dir):
    if workflow_file.endswith('.json'):
        print(f"\n=== {workflow_file} ===")
        issues = audit_workflow_nodes(os.path.join(workflow_dir, workflow_file))
        for issue in issues:
            print(f"  {issue}")
```

Action Items:

- [ ] Run audit on all workflows
- [ ] Create detailed issue report with node-by-node fixes
- [ ] Prioritize fixes by impact

Output: node_compatibility_report.md

Step 2.2: Update Critical Nodes

Time: 12 hours

Start with CRITICAL workflows (master agent, data extraction):

For each problematic node:

1. **Backup the workflow**
   ```
   Export workflow → Save as workflow_name_backup.json
   ```
2. **Update in UI:**
   - Open workflow in n8n
   - Find problematic node
   - Delete it
   - Add the new/replacement node
   - Reconnect wires
   - Test locally
3. **Code Node Migration (Example):**
   OLD (Function node v1):
   ```
   return [
     {
       json: {
         result: $input.first().json.data.map(x =&gt; x * 2)
       }
   ```

```
  }
];
```

NEW (Code node v1.119.2):

```
// Code node now supports async/await
const input = $input.first().json;
return [{
  json: {
    result: input.data.map(x =&gt; x * 2)
  }
}];
```

4. **HTTP Request Migration (Example):**
   OLD v1:

```
{
  "requestMethod": "GET",
  "url": "https://api.example.com/data",
  "authentication": "bearerToken",
  "nodeCredentialType": "httpHeaderAuth"
}
```

   NEW v2 (1.119.2):

```
{
  "method": "GET",
  "url": "https://api.example.com/data",
  "authentication": "predefinedCredentialType",
  "nodeCredentialType": "httpHeaderAuth"
}
```

Checklist for each CRITICAL workflow:

- [ ] Backup original
- [ ] Identify all problematic nodes
- [ ] Update each node
- [ ] Test with sample input
- [ ] Verify output matches original
- [ ] Export and save updated version

Step 2.3: Fix HIGH Priority Workflows

Time: 8 hours

Repeat Step 2.2 for all HIGH priority workflows (secondary extraction, classification, etc.)

Workflow List (Example HIGH):

- [ ] 02_Data_Extraction_Agent

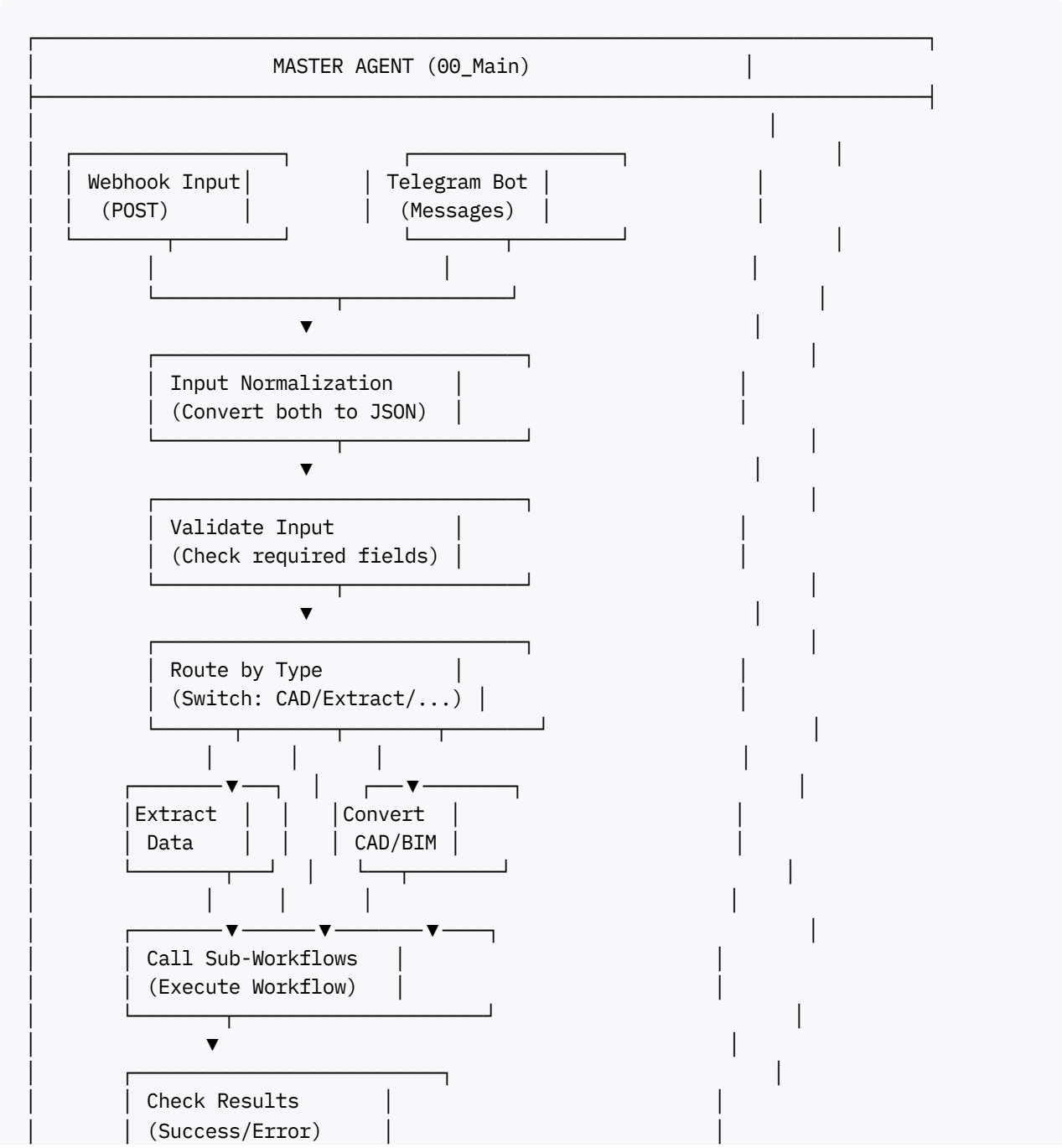
- [ ] 03_Material_Classification
- [ ] 04_Cost_Estimator
- [ ] 05_Carbon_Calculator
- [ ] 06_Quantity_Takeoff

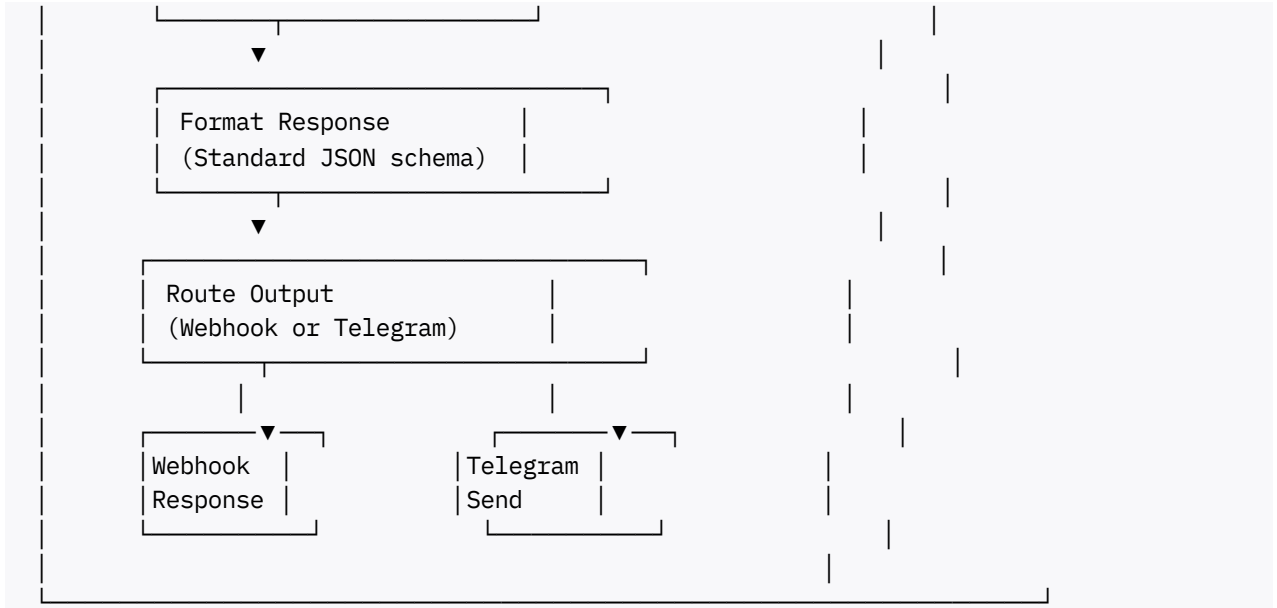
▯ Phase 3: Create Master Agent Workflow (Days 6-8)

Step 3.1: Design Master Agent Architecture

Time: 2 hours

Create visual architecture:





Step 3.2: Create Master Agent Workflow from Scratch

Time: 6 hours

In n8n UI, create new workflow: "00_Master_Agent"

Step 1: Add Triggers (both)

```
Node 1: Webhook Trigger
- Type: n8n-nodes-base.webhook
- Method: POST
- Path: /construction-ai
- Response Mode: "When last node finishes"
- Authentication: "None" (or API Key)

Node 2: Telegram Trigger
- Type: n8n-nodes-telegram
- Credential: Telegram Bot
- Listen on: both messages and file uploads
```

Step 2: Input Normalization (Function)

```
Node 3: Input Normalizer (Code)
- Type: n8n-nodes-base.code
- Language: JavaScript

Code:
──────────────────────────────────────────────────────────────────
const nodeType = $node.current.name; // Which trigger?

let normalizedInput = {
  source: "",
  requestType: "",
```

```
  fileExtension: "",
  file: null,
  data: null,
  telegram_user_id: null,
  telegram_message_id: null
};

// If Telegram trigger
if (nodeType === "Telegram Trigger") {
  normalizedInput.source = "telegram";
  normalizedInput.telegram_user_id = $json.from.id;
  normalizedInput.telegram_message_id = $json.message_id;

  if ($json.document) {
    normalizedInput.fileExtension = '.' + $json.document.mime_type.split('/')[1];
    normalizedInput.file = {
      url: $json.document.telegram_file_id,
      name: $json.document.file_name || "telegram_file",
      type: $json.document.mime_type
    };
  }

  if ($json.text) {
    normalizedInput.data = { text: $json.text };
    normalizedInput.requestType = parseCommand($json.text);
  }
}

// If Webhook
if (nodeType === "Webhook") {
  normalizedInput.source = "webhook";
  normalizedInput.requestType = $json.requestType || "";
  normalizedInput.fileExtension = $json.fileExtension || "";
  normalizedInput.file = $json.file || null;
  normalizedInput.data = $json.data || null;
}

function parseCommand(text) {
  if (text.includes("/extract")) return "extract_data";
  if (text.includes("/convert")) return "convert";
  if (text.includes("/classify")) return "classify";
  return "unknown";
}

return [{ json: normalizedInput }];
```

Step 3: Validation (Set + If)

```
Node 4: Validate Input (Set)
- Set JSON output schema:
{
  "source": "{{ $json.source }}",
  "requestType": "{{ $json.requestType }}",
  "fileExtension": "{{ $json.fileExtension }}",
```

```
  "file": "{{ $json.file }}",
  "validated": true
}

Node 5: Check Validation (If)
- If: requestType is NOT empty
- Else: Send error response (see error handling)
```

Step 4: Route by Type (Switch)

```
Node 6: Route Request (Switch)
- Property to evaluate: $json.requestType

Cases:
  "extract_data" → Path 1
  "convert" → Path 2
  "classify" → Path 3
  "cost_estimate" → Path 4
  "carbon_calculate" → Path 5
  ... etc
  Default → Error path
```

Step 5: Call Sub-Workflows

```
Node 7a: Execute Data Extraction (Extract Data path)
- Type: n8n-nodes-base.executeWorkflow
- Select Workflow: "02_Data_Extraction_Agent"
- Pass input: $json

Node 7b: Execute CAD Converter (Convert path)
- Type: n8n-nodes-base.executeWorkflow
- Select Workflow: "01_Convert_CAD_BIM"
- Pass input: $json

... (create similar nodes for each workflow)
```

Step 6: Merge Results

```
Node 8: Merge Results (Merge)
- Type: n8n-nodes-base.merge
- Combine all sub-workflow outputs
- Mode: "Merge by key"
```

Step 7: Format Response

```
Node 9: Format Response (Set)
- Create standard response schema:
{
  "success": true,
  "data": "{{ $json }}",
  "message": "Operation completed",
```

```
  "timestamp": "{{ now() }}",
  "executionId": "{{ $execution.id }}"
}
```

Step 8a: Webhook Response

```
Node 10a: Send Webhook Response (If source == "webhook")
- Type: n8n-nodes-base.respondToWebhook
- Response: $json (from Step 9)
- Status Code: 200
```

Step 8b: Telegram Response

```
Node 10b: Send Telegram Message (If source == "telegram")
- Type: n8n-nodes-telegram.telegramApi
- Method: sendMessage
- Chat ID: {{ $json.telegram_user_id }}
- Text: Format response as text
```

```
Node 10c: Send Telegram Document (If response has file)
- Type: n8n-nodes-telegram.telegramApi
- Method: sendDocument
- Chat ID: {{ $json.telegram_user_id }}
- File URL: {{ $json.data.file_url }}
```

Step 9: Error Handler (connect to all sub-workflows)

```
Node 11: Global Error Handler (attached to all execution nodes)
- Catch errors from all sub-workflows
- Format error:
{
  "success": false,
  "error": "{{ $error.message }}",
  "errorType": "{{ $error.type }}",
  "timestamp": "{{ now() }}"
}
- Send response via appropriate channel (Webhook/Telegram)
```

Validation Checklist:

- [ ] All 9+ core paths connected (no dead ends)
- [ ] Error handler on every sub-workflow call
- [ ] Both Webhook and Telegram responses implemented
- [ ] Test with sample webhook call
- [ ] Test with sample Telegram message
- [ ] Test error scenarios (invalid request, file not found, etc.)

Output: Master agent workflow active and tested

Step 3.3: Create Sub-Workflow Wrappers (for Master Agent compatibility)

Time: 4 hours

Ensure every sub-workflow:

1. Accepts normalized input from Master Agent
2. Returns standardized output
3. Has proper error handling
4. Handles both webhook and Telegram sources

For each sub-workflow (02_Data_Extraction, 03_Classification, etc.):

Add at START:

```
Node 1: Receive Input (Set)
- Accept input from Master Agent
- Ensure it has required fields
- Set default values if missing
```

Add at END:

```
Node N: Format Output (Set)
- Create standard response:
{
  "success": true,
  "result": {{ actual result data }},
  "metadata": {
    "workflow": "02_Data_Extraction",
    "processedAt": "{{ now() }}",
    "inputSource": "{{ $json.source }}"
  }
}
```

Add Error Handler:

```
Node N+1: Error Handler
- Catch any errors in workflow
- Return:
{
  "success": false,
  "error": "{{ $error.message }}",
  "workflow": "02_Data_Extraction"
}
```

Update all 45+ workflows with this pattern.

Phase 4: Integration & Testing (Days 9-15)

Step 4.1: Test Each Sub-Workflow Individually

Time: 12 hours (1.5 hours per workflow)

For each of the 30 most important workflows:

1. **Create test case**

```
{
  "source": "webhook",
  "requestType": "extract_data",
  "fileExtension": ".pdf",
  "file": {
    "url": "https://example.com/test.pdf",
    "name": "test.pdf",
    "type": "application/pdf"
  }
}
```

2. **Manual trigger** in n8n with test input
3. **Verify output**
   - success field is true/false
   - data field contains expected result
   - metadata field present
   - error field present if failed
4. **Check for errors**
   - Node type issues
   - Missing parameters
   - API authentication problems
5. **Document issues** and create fix tickets

Test Matrix:

```
Workflow Name | Test Input | Expected | Actual | Status | Issues
──────────────────────────────────────────────────────────────────
01_Convert_DWG | sample.dwg | coordinates | ✓ | PASS | None
02_Extract_Data | sample.pdf | text extracted | ✓ | PASS | None
... | ... | ... | ... | ... | ...
```

Step 4.2: Test Master Agent End-to-End

Time: 8 hours

Test Scenario 1: Webhook Input - Extract Data

```
curl -X POST http://localhost:5678/webhook/construction-ai \
  -H "Content-Type: application/json" \
  -d '{
    "requestType": "extract_data",
    "fileExtension": ".pdf",
    "file": {"url": "https://example.com/spec.pdf"}
  }'

Expected Response:
{
  "success": true,
  "data": { "result": "..." },
  "message": "Operation completed"
}
```

Test Scenario 2: Webhook Input - Convert CAD

```
curl -X POST http://localhost:5678/webhook/construction-ai \
  -H "Content-Type: application/json" \
  -d '{
    "requestType": "convert",
    "fileExtension": ".dwg",
    "file": {"url": "https://example.com/design.dwg"}
  }'
```

Test Scenario 3: Telegram Input - Extract Data

```
Send message to bot: "/extract"
+ Attach file (PDF)

Expected Response:
Bot replies with extracted data + file if applicable
```

Test Scenario 4: Error Handling

```
# Missing required field
curl -X POST http://localhost:5678/webhook/construction-ai \
  -H "Content-Type: application/json" \
  -d '{"requestType": ""}'

Expected Response:
{
  "success": false,
  "error": "Missing required field: requestType"
}
```

Test Checklist:

- [ ] Webhook POST returns JSON with success/error
- [ ] Telegram sends message back
- [ ] All requestTypes routed correctly
- [ ] Error scenarios handled gracefully
- [ ] Response time < 5 seconds for most operations
- [ ] No undefined nodes or errors in n8n logs
- [ ] Data returned is correct format

Step 4.3: Load Testing

Time: 4 hours

Use Apache JMeter or Artillery to simulate concurrent requests:

```
# load-test.yml
config:
  target: "http://localhost:5678"
  phases:
    - duration: 60
      arrivalRate: 5

scenarios:
  - name: "Extract Data"
    requests:
      - post:
          url: "/webhook/construction-ai"
          json:
            requestType: "extract_data"
            file:
              url: "https://example.com/test.pdf"
```

Success Criteria:

- [ ] Handle 5-10 concurrent requests without errors
- [ ] Average response time < 3 seconds
- [ ] No timeout errors
- [ ] All responses have proper error handling

Step 4.4: Data Quality Testing

Time: 6 hours

For each extraction/processing workflow:

1. **Test with real data**
   - Sample PDF files
   - Sample CAD files
   - Sample Excel sheets
2. **Verify output accuracy**
   - Extract matches expected fields
   - Data types correct
   - No data loss or truncation
3. **Test edge cases**
   - Large files (>100MB)
   - Complex documents (many pages)
   - Unusual formats

Quality Checklist:

- [ ] Extraction accuracy > 95%
- [ ] No data loss on any test files
- [ ] Large files handled without timeout
- [ ] Output format consistent

Phase 5: Deployment & Documentation (Days 16-20)

Step 5.1: Create Deployment Runbook

Time: 2 hours

Deployment Steps:

1. **Backup production workflows**

```
# Export all workflows
n8n export --output workflows_backup_$(date +%Y%m%d).json
```

2. **Schedule maintenance window**
   - 2 AM UTC (low traffic time)
   - Announce to team 24 hours before
3. **Import master agent workflow**

```
n8n import --file 00_Master_Agent.json
```

4. **Import updated sub-workflows**

```
for workflow in 01_*.json 02_*.json 03_*.json; do
  n8n import --file "$workflow"
done
```

5. **Verify all workflows**
   - Check n8n UI that all imported successfully
   - Verify no error badges on workflows
   - Check webhook endpoints active
6. **Run smoke tests**
   - Test master agent with sample request
   - Test each major requestType
   - Check Telegram bot responding
7. **Monitor for 1 hour**
   - Check logs for errors
   - Monitor CPU/memory usage
   - Verify webhook calls being processed
8. **Rollback procedure (if needed)**

```
# Delete new workflows
n8n delete --id &lt;workflow_id&gt;
# Import from backup
n8n import --file workflows_backup_$(date +%Y%m%d).json
```

Step 5.2: Create User Documentation

Time: 4 hours

Document:

1. How to use webhook endpoint
2. How to use Telegram bot
3. Available requestTypes
4. Input/output schema examples
5. Troubleshooting guide
6. API rate limits

Example:

```
# Construction AI - N8N Workflows Guide

## Webhook Usage

### Endpoint
POST https://your-server.com/webhook/construction-ai

### Example Request
{
  "requestType": "extract_data",
  "fileExtension": ".pdf",
  "file": {
    "url": "https://example.com/document.pdf"
  }
}

### Available RequestTypes
- extract_data: Extract text/data from documents
- convert: Convert CAD/BIM files
- classify: Classify materials
- cost_estimate: Calculate costs
- carbon_calculate: Calculate carbon footprint
- quantity_takeoff: Extract quantities
...

## Telegram Bot Usage

### Commands
/extract - Extract data from document (attach file)
/convert - Convert CAD file (attach file)
/status - Check workflow status
...

### Limitations
- Maximum file size: 50MB
- Response time: 5-30 seconds
- Rate limit: 10 requests/minute per user
```

Step 5.3: Create Monitoring Dashboard

Time: 3 hours

Set up monitoring for:

- Workflow execution success rate
- Average response time
- Error frequency by type
- Webhook calls per hour
- Telegram messages processed

Use n8n's built-in analytics or connect to external monitoring (Grafana, CloudWatch).

Alerting Rules:

- Alert if success rate drops below 95%
- Alert if response time exceeds 10 seconds
- Alert if more than 10% errors
- Alert on any undefined node errors

Step 5.4: Operator Training

Time: 2 hours

Train team on:

1. How to update workflows safely
2. How to test new workflows
3. How to troubleshoot common issues
4. When to rollback vs. fix forward
5. Monitoring and alerting

Summary Timeline

```
Week 1:
  Day 1-2: Audit &amp;amp; Setup (10 hours)
  Day 3-5: Node Compatibility (28 hours)

Week 2:
  Day 6-8: Master Agent Creation (8 hours)
  Day 9-10: Sub-workflow wrapping (8 hours)

Week 3:
  Day 11-15: Testing &amp;amp; Validation (40 hours)

Week 4:
  Day 16-20: Deployment &amp;amp; Documentation (11 hours)

TOTAL: ~105 hours (~3-4 weeks, 1 engineer)
```

Delivery Checklist

Week 1:

- [ ] All workflows audited and issues documented
- [ ] Testing environment set up
- [ ] Integration map created

Week 2:

- [ ] All critical and high-priority workflows updated
- [ ] Master agent workflow created
- [ ] Sub-workflows wrapped for compatibility

Week 3:

- [ ] All 30+ major workflows tested individually
- [ ] Master agent end-to-end testing complete
- [ ] Load testing passed
- [ ] Data quality verified

Week 4:

- [ ] Production deployment completed
- [ ] Documentation created
- [ ] Monitoring configured
- [ ] Team trained

Rollback Plan

If critical issues arise post-deployment:

1. **Stop webhook trigger** (pause master agent)
2. **Alert team** of rollback
3. **Import backup workflows** from Day 1
4. **Verify systems operational** with old workflows
5. **Investigate root cause** with test environment
6. **Fix and retry** when confident

Support & Escalation

Issues during upgrade:

- Node compatibility: Check n8n changelog for v1.119.2
- API failures: Verify credentials and rate limits
- Performance: Check n8n server resources
- Data loss: Restore from backup immediately

Escalation contacts:

- N8N support: https://n8n.io/support
- Your team tech lead

- DevOps/Infrastructure team

This plan is comprehensive and achievable. Start with Phase 1 (Audit) this week, and follow the timeline to have a fully upgraded system within 3-4 weeks. Good luck!