International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 12 Issue: 12 | Dec 2025 www.irjet.net p-ISSN: 2395-0072

A Practical Evaluation of Agentic AI and No-Code Workflow Orchestration (n8n) for Adaptive Backend System Replacement

Nazifa Hajwane[1], Prof Ashok Yadav[2]

[1]Student, Department of Information Technology, Reena Mehta College of Arts, Commerce, Science & Management Studies, Mira Bhayandar West, Maharashtra 401101
[2]Prof and Head of department (Department of information technology), Reena Mehta College of Arts, Commerce, Science & Management Studies, Mira Bhayandar West, Maharashtra 401101, India

---

Abstract - *Traditional backend systems take time to build, are hard to maintain, and rely on rigid business rules written in code. This paper studies whether AI agents, combined with the no-code workflow platform n8n, can replace large parts of these traditional back ends systems. We compare two approaches: deterministic AI workflows and non-deterministic AI agents. The comparison is based on reliability, cost, latency, and ease of debugging. This study shows that AI agents can replace a significant portion of backend logic when used carefully with orchestration tools like n8n. However, the analysis also reveals key limitations related to scalability, predictability, and the need for structured, reactive prompt engineering. The findings highlight that while agented systems offer superior flexibility for complex and dynamic tasks, deterministic AI workflows remain more reliable and cost-effective for predictable backend processes.*

Key Words*: *AI Agents, Workflow Automation, n8n, LLMs, Backend Replacement, Multi-Agent Systems, Retrieval Augmented Generation (RAG), Cost Optimization, Reactive Prompting.

1. INTRODUCTION

Modern application development is progressively focusing on separating the front end of an application from the backend of the application [6]. The front end is easier to build using frameworks like react or low-code tools unlike the back-end infrastructure that is more complex to implement tasks such as user authentication, processing, business logic execution, and API interactions. These tasks require skilled developers because they require significant manual coding in core languages like Python or Java. The blend of Large Language Models (LLMs) and no-code orchestration plat forms like **n8n** gives an opportunity to fundamentally transform these systems. AI agents that are powered by LLMs help to interpret natural language instructions and translate them into executable workflows, interacting dynamically with APIs, databases, and authentication systems. This paper moves beyond a conceptual framework this way it establishes an evidence-based analysis that is focused on three key areas:

1. **Methodology:** It is used to describe measurable metrics to compare performance of AI agents and linear AI workflows [4].

2. **Implementation**: Offering detailed prototypes for the argentic systems like (Invoice Extraction, RAG Chat bot, Multi-Agent Orchestration) using n8n nodes, variable mapping, and JSON schema output.

3. **Critical Analysis:** The effect of dynamic selection of models on operational costs is analyzed critically, followed by the importance of structured prompt engineering for getting reliable results.

2. TECHNICAL FOUNDATIONS AND RELATED WORK

2.1. LLMs and Argentic AI

An LLM acts as the core processing unit, accepting an input, and generating an output. However, its practical utility is limited. Its true potential is unlocked when it is connected to tools such as databases, APIs, and file systems. An AI Workflow processes information sequentially, following a deterministic path. Each step is predefined and executed in order. This makes it predictable and easy to debug. (e.g., Step 1: Input Lead, Step 2: Research Lead, Step 3: Draft Email, Step 4: Send Email). Conversely, an AI Agent is a non-deterministic system capable of making autonomous decisions. It decides which tool to use, in what order, and how many times. These decisions are guided by a system prompt and memory. This flexibility makes agents powerful but also harder to control.

2.2. n8n as an Orchestration Layer

n8n is an open-source node-based platform that allows for the visual organization of back-end logic. It is the flexible middle ground through which the AI agent must connect with real-world systems [8]. Primary features include:

2.2.1. **Nodes**: These are graphical elements meant for application actions like Google Sheets or Slack, data transformation, API calls (HTTP Request), and some AI tasks.

2.2.2. **Triggers**: These are what initiate the workflow process. These triggers include Web hook, Schedule, App Event, Manual Trigger.

2.2.3. **Data Flow**: Data is shared between nodes through JSON (JavaScript Object Notation) key-value pairs. Key features of JSON key-value pairs include dynamic variables, for (e.g., {{ $json.message. content }}) which enable flexibility by accessing previous node outputs.

2.3. Core Backend Components Replaced by AI

AI agents can fully or partially replace several critical backend elements by translating natural language into an executable logic with the help of:

2.3.1. **Data Management**: Agents can convert user commands into dynamic SQL/NoSQL queries to enable data management using the CRUD operations [5].

2.3.2. **Business Logic Execution:** Agents remove fixed conditional statements in business rules by automatically creating decision trees through user descriptions (e.g., applying a discount if an order value exceeds a threshold) [6].

2.3.3. **API Integration:** These agents make API integrations easier by defining HTTP requests that are automatically formed by the agents while integrating APIs

3. METHODOLOGY: COMPARING AI WORKFLOWS AND AI AGENTS

To progress from anecdotal reports, it is important for us to employ a methodology to evaluate and compare choices between non-deterministic AI Agents and linear, deterministic AI Workflows, specifically analyzing reliability, cost, and debugging efficiency [4].

Table-1: Comparative Analysis of Deterministic AI Workflows and Non-Deterministic AI Agents

Metric	AI Workflow (Deterministic)	AI Agent (non-deterministic)	Rationale
Reliability/Consistency	High. Follows strict guardrails (Step 1 -> 2 -> 3).	Lower. Prone to hallucination or inconsistent tool-calling logic.	Linear flows are less prone to deviation and provide consistent outputs.
Cost Efficiency	High. LLM is called only when necessary (e.g., once per content generation).	Lower. LLM is called repeatedly (hits its "brain") to decide next steps.	Eliminating decision-making API calls results in significant token savings.
Debugging	Easy. Failures are pinpointed to a single node or variable mapping.	Difficult. Errors often result from complex system prompt failures or data transfer issues between agent/tool.	Visibility into the linear path simplifies error isolation.
Adaptability/Flexibility	Low. Requires manual structural changes for new logic.	High. Can adapt to complex, multi-step queries by calling tools dynamically.	Agents are necessary when the process is unpredictable (non-deterministic).

The results presented in Table-1 demonstrate that AI Workflows are preferred for predictable, sequential tasks (e.g., lead qualification, invoice processing) due to superior cost and reliability. AI Agents are reserved for complex queries requiring dynamic tool selection, external knowledge retrieval (RAG), or layered decision-making. The key finding is that AI Workflows are preferable for deterministic, sequential tasks such as lead qualification or invoice processing for their cost-effectiveness and reliability. AI Agents are reserved for complex queries involving dynamic tool selection, or retrieving external knowledge (RAG), or multi-layer decision-making.

4. IMPLEMENTATION AND DETAILED PROTOTYPES

4.1 Prototype-1: AI-Powered Invoice Information Extraction Workflow

This prototype is used to extract structured data from unstructured PDF invoices without the need for predefined templates. The system successfully handled invoices with different formats [8].

Table-2: Node-Level Workflow Configuration for AI-Powered Invoice Information Extraction

Node	Operation	Configuration & Data Flow	Source Detail
1. Google Drive Trigger	On File Created	Watches a designated Google Drive folder for new PDF files. Output: File ID.	Detecting new PDF files in Google Drive.
2. Google Drive	Download File	Dynamic input: Uses the File ID variable ({{$json.id}}) from Node 1 to download the file. Output: Binary Data.	Downloa d File
3. Extract from File	Extract Text from PDF	Input: Binary Data from Node 2. Output: Raw Text Content.	Extractin g text from the file.
4. AI Informatio n Extractor	Attribute Descriptio n	*Input:* Raw Text Content from Node 3. *Attributes Defined:* Eight specific required fields (Invoice Number, Client Name, Total Amount [as String], Due Date [as Date], etc.).	Using an AI model to identify key fields.
5. Google Sheets	Append Row	Dynamic Input: Maps the structured JSON output fields from Node 4 (e.g., {{$json.output["Invoic e Number"]}}) directly to the Google Sheet columns.	Storing results in Google Sheets.

The results presented in Table-2 demonstrate that the workflow successfully processed invoices from different vendors, including ABC Tech Solutions and XYZ Enterprises. The AI model accurately extracted all required fields into a consistent JSON structure.

4.2 Prototype-2: Retrieval Augmented Generation (RAG) Chat bot

This implementation demonstrates a core RAG pipeline which is essential for providing agents with real-time, domain-specific context external to the LLM's training data. This way the prototype improves chat bot accuracy by providing external context. Documents are stored in a vector database. When a user asks a question, the agent retrieves relevant information before generating an answer. This prevents hallucination and improves factual accuracy.

Table-3: Retrieval Augmented Generation (RAG) Pipeline Configuration and Data Flow

Phase/Node	Operation	Configuration & Data Flow
Ingestion	Pine Cone Vector Store	Ingestion involves splitting documents into chunks, transforming text into vectors using an embeddings model (e.g., OpenAI text-embedding-3-small), and inserting them into a Pine Cone index namespace (e.g., 'FAQ').
Agent Core	AI Agent	Chat Model (e.g., Claude 3.5 Sonnet via Open Router). System Prompt (Optional): Defines role and tool use. Input: User query ({{ $node["Chat Trigger"].json["text"] }}).
Tool	Pine Cone Vector Store	Tool Name: "Knowledge Base." Retrieves documents. Configured with matching index, FAQ namespace, and the same embeddings model used during ingestion.

The results presented in Table-3 demonstrate that when asked about a 'warranty policy,' relevant document chunks were retrieved from the vector database. and generated an accurate response using the retrieved context. The response correctly included details about coverage and exclusions. This way it confirms context enrichment and tool use autonomously.

4.3 Prototype-3: Multi-Agent Orchestration for Delegation

This architecture uses an Orchestrator Agent to distribute complex queries across specialized Subagents (Child Workflows), thereby solving the problem of specialization and scalability.

4.3.1. **Orchestrator Agent (Parent):** The Orchestrator does not perform tasks itself. It only delegates them. It uses minimal tools, primarily being the n8n Workflow Tool nodes.

4.3.2. **Sub-Agent (Child Workflow):** A dedicated, specialized workflow (e.g., "Email Agent").It would start with an Execute Workflow Trigger and receive the query of the Orchestrator via the query field, which would perform some specific actions, such as sending an email using a Gmail tool.

4.3.3. **Specialization Benefit:** Simple prompts improve reliability and allow low-cost models to be used for basic tasks while advanced models are reserved for complex reasoning.

5 RESULTS AND PERFORMANCE EVALUATION

5.1 Cost Optimization through Dynamic Model Selection

An argentic system was implemented for complex tasks to dynamically select the best LLM required based on the user's intent, to maximize cost efficiency without degrading its quality.

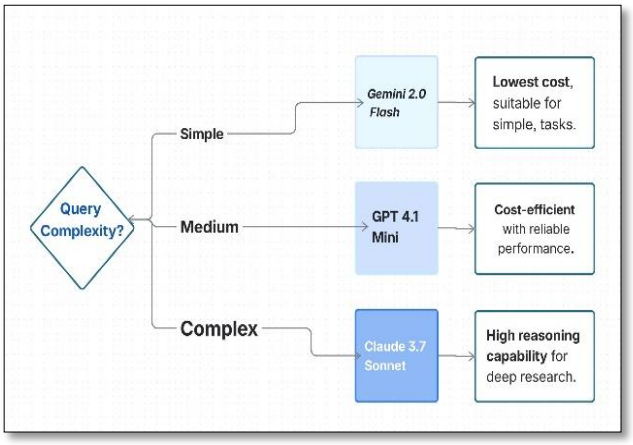

Figure 1: Dynamic LLM Selection Strategy Based on Query Complexity

Figure 1 demonstrates the core LLM of the main agent is updated dynamically using a Model Selector Agent which classifies the request and gives only the model's name as output. This practice-as in here, the system selects different LLMs based on task complexity for example - simple tasks will use low-cost models (like Gemini 2.0) and extraordinarily complex tasks will use high-reasoning models (like Claude) . This way a Model Selector Agent reduces the overall cost.

5.2 Advanced API Integration and Asynchronous Processing

n8n simplifies API integration by allowing direct cURL imports. For long-running tasks, the workflow uses asynchronous polling to wait for task completion. Reliable backend automation requires integrating APIs from third-party sources. Common use cases include Data scraping (Tavly, Brave Search) and Media generation (Runway, OpenAI Image Generation)

5.2.1. **HTTP Requests and cURL:** n8n makes it easier to be set up with complex APIs allowing developers to import a cURL command into n8n and see immediate configuration of the HTTP method-POST/GET, endpoint URL, and body/headers parameters. Without the need for meaningful manual coding that usually goes into API wrappers.

5.2.2. **Polling Asynchronously:** For long-running media generation tasks, such as creating a 10-second video via Runway or scraping a full website via Fire crawl, the workflow needs to be able to process asynchronous responses. First, the system sends request Node 1, returning an Execution ID. At this point, the workflow enters a polling loop: it waits (e.g., 5 seconds) and then checks the status of the Execution ID until its status changes from "running" to "succeeded". If an application has a long-running task, then it should implement a method that returns an Execution ID for subsequent polling of that execution status. This polling loop ensures the workflow receives the completed result without timing out or breaking the connection.

6 DISCUSSION AND CRITICAL ANALYSIS

6.1 Reactive Prompt Engineering for Robustness

The reliability of an AI agent is directly proportional to the quality of its system prompt [7, 9]. Reactive prompting is preferred over proactive prompting.

6.1.1 **Reactive Approach:** The agent is initiated with a minimal, generic system prompt, such as "You are a helpful assistant." Detailed instructions are added only when errors or other undesirable behavior are found during testing. For instance, if an agent fails to check the contact database before sending an email, the reactive fix is to insert a rule: “Always check the contacts database before sending an email.

6.1.2 **Structured Output Parsing:** It is enforced using strict JSON schema. For reliable downstream automation (e.g., mapping output to a Gmail node), the agent's output must be predictable. This is achieved using a Structured Output Parser node configured with a strict JSON schema. This ensures that variables like to, subject, and body are consistently available as separate fields, simplifying variable mapping and preventing failures.

6.2 Limitations of No-Code and Scaling Challenges

While n8n is highly effective for building robust MVPs (Minimum Viable Products) and internal automations tools. Scaling to enterprise-level challenges in high-volume production systems.

6.2.1 **Performance and Data Volume:** When dealing with massive data sets or complicated multi-agent flows may cause latency. In cases of truly massive data processing a hybrid approach may be needed

for Python scripts for heavy data processing and n8n for orchestration

6.2.2 **Tool Complexity (MCP Servers):** Although the Model Context Protocol (MCP) enables dynamic tool discovery and execution tools (e.g., Brave Search, Air Table) without manual API setup, limitations still exist. However, multi-step database actions like (first retrieving a Base ID, then a Table ID, then listing records) still require complex prompts revealing that the full autonomy for relational queries is not yet achieved.

6.2.3 **Human in the Loop (HITL) Agents:** The existing architecture makes it impossible for HITL systems to be easily integrated as agent tools. Though the HITL workflows function perfectly as linear workflows, decision-making steps, using a human-approval step as a tool inside a more complex, delegating agent often causes it to stall or not pull in the human's feedback dynamically.

7. CONCLUSION

AI agents combined with n8n offer a powerful alternative to traditional backend systems. Rigid, hard-coded logic can be replaced with adaptive intelligence. Real-world prototypes successfully validate automated Invoice extraction, RAG chatbots and multi-agent orchestration. Successful deployment critically depends on careful architectural decisions regarding AI Workflow vs. AI Agent and rigorous methodological practices such as reactive prompt engineering, structured output parsing and systematic data organization methods which are designed for reliability, predictability, and usability of large language model outputs. As LLMs and platforms like n8n continue to evolve, the distinction between a system designer (who orchestrates AI-powered logic) and a code writer will become increasingly pronounced, this innovation will cause backend development to shift from writing code to designing intelligent systems.

REFERENCES

[1] S. Bandari, Replacing AI Agents for Backend, 2024. [Online]. Available: https://www.researchgate.net/profile/Sai-Bandari/publication/392678233_Replacing_AI_Agents_for_Backend/links/6854322224267473b778e856/Replacing-AI-Agents-for-Backend.pdf

[2] F. Huang, "Data processing," in Encyclopedia of Big Data, L. Schintler and C. McNeely, Eds. Cham, Switzerland: Springer, 2019. doi: 10.1007/978-3-319-32001-4_314-1.

[3] M. Land and M. Crnkovic, "Existing approaches to software integration and a challenge for the future," in Proc. Int. Conf. Software Engineering Research and Practice (SERP), Las Vegas, NV, USA, 2002, pp. 1–7. [Online]. Available: https://www.semanticscholar.org/paper/60aa78b942b488456e529834ef6b95d9913e7ccb

[4] W. T. Lehto and M. S. Liddell, "A study on boilerplate code in modern programming languages," IEEE Trans. Software Engineering, vol. 45, no. 6, pp. 845–856, Jun. 2019, doi: 10.1109/TSE.2019.2899047.

[5] P. S. Lee, M. H. Cho, and Y. G. Kim, "Efficient debugging of backend logic in distributed systems," IEEE Trans. Software Engineering, vol. 46, no. 5, pp. 673–688, May 2020, doi: 10.1109/TSE.2020.2990485.

[6] [6] A. J. Smith and L. S. Kumar, "The role of DevOps in enhancing backend system efficiency," IEEE Trans. Software Engineering, vol. 45, no. 8, pp. 985–996, Aug. 2020, doi: 10.1109/TSE.2020.2991498.

[7] [7] T. B. Brown et al., "Language models are few-shot learners," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 1877–1901, doi: 10.5555/3454287.3454752.

[8] [8] B. Ho, "n8n: A free and open workflow automation tool," n8n Blog, 2020. [Online]. Available: https://n8n.io/blog/introducing-n8n

[9] [9] J. Smith, "AI agents in modern software systems: A comprehensive review," IEEE Access, vol. 8, pp. 104356–104375, 2020, doi: 10.1109/ACCESS.2020.2994365.

[10] A. Nguyen and J. Lee, "Server-sent events for real-time web applications," IEEE Trans. Web Engineering, vol. 16, no. 2, pp. 101–113, 2020, doi: 10.1109/TWE.2020.2980165.

[11] A. Williams and B. Patel, "Optimizing API queries with GraphQL," IEEE Trans. Software Engineering, vol. 43, no. 10, pp. 1031–1042, 2018, doi: 10.1109/TSE.2018.2798494.