International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

NODOMO: A Browser-Native, AI-Powered Visual Workflow Automation Platform with Intelligent Node Recommendation and Zero-Server Architecture

Mr. Yash Tejraj Thorat[1], Mr. Sai Sunil Varade[2], Mr. Rohit Kishor Patil[3], Mr. Shubham Dnyaneshwar Patil[4], Mr. Ujwal Kamalakar Patil[5], Dr. Mayur N. Agrawal[6]

Abstract

Automating business workflows sounds simple until you actually try to do it. Most people hit a wall the moment their automation needs something slightly custom a conditional check an API call with a non standard header or an AI model in the middle of the pipeline. Tools like Zapier work fine for basic stuff but fall short fast and n8n is while powerful but needs Docker and server setup just to get started which is not something a nont echnical person can realistically do on their own.

We built NODOMO to sit in the space between these two. It is a workflow automation platform that runs completely in the browser no server, no installation, no monthly subscription. Users drag nodes onto a canvas and connect them to build automations. There is also a natural language interface where you just describe what you want and the AI generates the workflow for you.

The system has over 50 node types covering AI providers like Groq, Gemini, OpenAI and Anthropic plus communication, data, logic and content nodes. Under the hood the execution engine uses BFS traversal with Kahn's algorithm for topological ordering and DFS for cycle detection. We also built a three stage recommendation engine that suggests the next node based on rule matching. Jaccard similarity with template workflows, and a Random Forest classifier as a fallback. API keys never leave the browser they are encrypted with AES-GCM 256-bit using the Web Crypto API.

The tech stack is Next.js 15, TypeScript strict mode, React Flow v12, Zustand v5, TanStack Query v5, Tailwind CSS v4 and Zod. It is deployed on Vercel. This paper explains the architecture, the algorithms, and the design decisions we made along the way.

Keywords: workflow automation, AI agents, node-based editor, no code platform, browser native, agentic AI, BFS execution, DAG, recommendation engine, random forest, AES-GCM, Next.js, React Flow

INTRODUCTION

A. Where We Started

When we first started thinking about this project the ques-tion we kept coming back to was pretty simple why is automa-tion still so hard for regular people? The technology is clearly there. LLMs can reason through complex tasks basically every SaaS product has an API and visual builders have existed for

years. But if you ask someone without a software background to build even a moderately complex automation, they struggle. They either give up because the tool is too technical or they end up paying a lot for something like Zapier that works fine for simple cases but breaks down the moment requirements get more complicated.

McKinsey's 2024 AI survey found that 88% of organiza-tions are using AI in at least one function but only around one third have managed to scale it across the enterprise [1]. That gap exists for a lot of reasons but one of them is definitely tooling. The tools that are easy to use are limited and the tools that are powerful are hard to use. Nobody has really solved both at the same time.

What made it worse for us as CS students is that the major automation platforms were all built before LLMs were useful. Zapier launched in 2011 and n8n came out in 2019. These tools were designed around API connectors and simple trigger-action pairs and AI was added later as just another connector. So things that should be straightforward like managing context across multiple LLM calls in a workflow or generating a work-flow from a plain language prompt are either not supported or implemented in a hacky way.

There is also a research angle here. Li et al. in AutoFlow

[2] point out that LLM-based automation relies heavily on manually designed agentic workflows and designing those workflows takes a lot of human effort. There has been good work on automating the generation of workflows but less work on how the actual interface for building and modifying workflows should look. That is the problem we focused on.

B. What We Built

NODOMO is a visual workflow builder that runs entirely in the browser. You drag nodes onto an infinite canvas, connect them with edges and run your automation. There is no backend handling the execution, no database storing your workflows and no server receiving your API keys. Everything stays on your device.

The name NODOMO stands for Node-based Domain Ori-ented Modular Orchestration. But the idea is that every step in your automation is a self contained node with typed inputs and outputs and you organize these nodes by connecting them into a directed acyclic graph.

The thing that sets NODOMO apart is not any one feature in isolation. It is free and open source. It works without a server.

C. Main Contributions

Here is what this paper specifically contributes:

- A browser native workflow automation architecture with no backend dependency for workflow creation, storage or execution.
- A BFS execution engine with DFS cycle detection, Kahn's topological sort and step level retry logic all running client-side.
- A three stage hybrid recommendation engine : rule matching then Jaccard similarity against templates then Random Forest classification.
- AES-GCM 256-bit API key encryption using the native Web Crypto API with no server transmission of creden-tials at any point.
- A Hierarchical Context Parsing Tree (HCPT) for man-aging LLM context across multi-node AI chains without blowing the context window.
- A library of 50+ node types covering AI, data, logic, communication, content and utilities.

D. Paper Organization

Section II covers related work. Section III goes into the specific problems we are solving. Section IV describes the architecture. Section V explains the core algorithms. Section VI covers technology choices. Section VII goes through use cases. Section VIII is about security. Section IX is future work. Section X concludes.

LITERATURE SURVEY

A. Agentic Workflow Research

There has been a fair amount of academic work on LLM-based workflow automation in the last two years and it is worth understanding where NODOMO sits relative to that.

AFLOW [4], which came out at ICLR 2025, frames work-flow generation as a search problem and uses Monte Carlo Tree Search to find good workflow configurations. The results look promising on benchmarks but the compute cost is high and it is not suitable for real time interactive use. That is not really what we were going for anyway. NODOMO is about the human building the workflow with AI assistance not fully autonomous workflow synthesis.

AutoFlow [2] is closer to what we do. It has an LLM agent that generates workflow specifications from a high-level goal. The main difference is that in AutoFlow the generated workflow just runs programmatically, while in NODOMO it appears on a visual canvas that the user can inspect, modify, and run interactively. The visual feedback loop matters a lot for usability.

WorkflowLLM [3] builds a benchmark for testing LLM capability on workflow tasks and finds that current models are still not great at long horizon planning and correct tool chaining. This is actually one of the reasons we did not go fully AI driven. Our hybrid approach is AI generates a starting point then human refines it visually and more robust than asking an LLM to get everything right on the first try.

Workflow-R1 [5] proposes a reinforcement learning ap-proach where each node addition is treated as an action in a decision process. Interesting direction for fully automated synthesis, but again, for our use case where a real user is building the workflow the interactive model makes more sense.

B. Multi-Agent Frameworks

AutoGen [7] from Microsoft is probably the most well-known multi-agent LLM framework right now. It sets up agents that can use tools and pass messages between them-selves to complete tasks collaboratively. NODOMO's AI agent node is conceptually similar it is an LLM with a set of tools available to it where the tools are the downstream nodes. The difference is that in NODOMO this is all configured visually rather than in code.

API-Bank [9] is a benchmark for LLM tool use and finds that accurate tool selection and parameter filling are still hard problems even for capable models. We felt this in practice too. The AI copilot in NODOMO does well at suggesting which node to add next but getting the full configuration of complex nodes right automatically is still something users often have to fix manually.

C. Visual and Low-Code Approaches

Low-code LLM [11] is probably the most directly related work to NODOMO's visual design. They built a graphical interface for composing LLM pipelines with nodes and found that visual representation significantly reduces errors in com-plex multi-step setups. That matches what we observed during development once you can see the full pipeline on the canvas it is much easier to spot logical errors than when you are reading workflow configuration files.

Graph of Thoughts [10] extends chain of thought prompting

[8] to non linear graph structures, which maps well to the DAG based execution model we use. We are not implementing Graph of Thoughts specifically but the underlying idea that graph structured representations help with complex reasoning tasks is something we agree with.

DSPy [6] is a programming model for LLM pipelines that compiles declarative specifications into optimized prompt chains. The compilation step is somewhat analogous to how NODOMO's AI generation converts a natural language de-scription into a workflow graph. The key difference is that DSPy is a developer tool while NODOMO aims to be usable by non-developers.

D. Graph Algorithms

On the algorithms side, NODOMO's execution engine is grounded in classical CS. Kahn's algorithm [13] gives us topological ordering for the execution sequence. DFS based cycle detection [14] prevents invalid workflow graphs. BFS traversal handles the actual execution. Breiman's Random Forest [12] is the ML component in the third stage of our recommendation engine. The Bitap algorithm [15], via Fuse.js, powers the fuzzy search in the command palette.

PROBLEM STATEMENT

A. The Actual Barriers

We want to be specific here because "automation is hard" is too vague. The barriers that actually stop people are more concrete than that.

Technical knowledge gap. Even on Zapier, once you need anything non standard a custom API header, a dynamic field from a previous step or a conditional branch the interface gets confusing fast. Users who are not developers often cannot debug these situations on their own. For n8n it is even more stark : you need Docker, a server, a domain, SSL setup and a database just to run the thing. Most non technical people cannot do that.

Cost. Zapier's paid tiers range from about $20 to over $600 per month depending on task volume. Make.com charges per operation. Enterprise platforms like UiPath are in a completely different price range. For a student, a freelancer, a small startup or a researcher these costs are just not viable. There should be a free, powerful option and currently there is not.

AI was added as an afterthought. This one bothers us the most. Zapier, n8n, Make.com they were all designed before LLMs were useful. When AI became important these platforms added AI connector nodes but the underlying archi-tecture was never designed for AI native workflows. Context management across multiple LLM calls, natural language workflow generation, AI assisted debugging none of these are first class features anywhere. They are bolt-ons.

Vendor lock-in. Workflows on proprietary platforms are stored in closed formats. You cannot export them in a readable way, version control them or move them between platforms without rebuilding from scratch. If the platform changes pricing or shuts down your automations are gone.

Privacy. Every major cloud automation platform processes your data on their servers. If your workflow handles customer data, internal business data, or sensitive API responses all of that is passing through a third party's infrastructure. For a lot of organizations this is a real compliance and security concern not just a theoretical one.

Debugging is painful. When something fails most plat-forms show you a raw error code or a JSON response from the failed API call. That means nothing to someone who is not a developer. There is no system that looks at what you were trying to do and explains in plain language what went wrong and how to fix it.

B. The Research Gap

Most of the academic work we reviewed focuses on au-tomating workflow generation or benchmarking LLM work-flow capability. Less work has focused on the interaction design side how a visual interface should be designed to make complex AI workflows accessible to people who are not software engineers. NODOMO is specifically trying to address that gap.

SYSTEM ARCHITECTURE

A. Running Everything in the Browser

The most important architectural decision we made is that NODOMO runs entirely in the browser. No backend server processes your workflow data. No database stores your workflows remotely. No server side runtime executes your automation steps. It all happens client side.

This choice was partly philosophical and partly practical. If there is no server, there is no data leakage. Data privacy is guaranteed by the architecture itself not by a privacy policy. It means the platform can be deployed as a static site on Vercel and scale to any number of users at near zero cost. It also means the whole thing works offline for workflows that do not call external APIs.

There is a trade off. Things like incoming webhooks or scheduled background jobs need at least a serverless function endpoint. We handle that through Vercel Edge Functions while keeping everything else storage, canvas state, execution logic on the client side.

B. Architecture Layers

We structured the system into five layers that each handle a distinct responsibility.

The **Presentation Layer** is the Next.js 15 App Router application with React 19. The landing page is statically generated for load performance. The workspace itself is client-side rendered since it is a fully interactive canvas application with no server side data requirements. The workspace has the React Flow canvas, a left sidebar with the node library, a right panel for node configuration, a bottom console for execution logs and a command palette.

The **State Management Layer** uses four Zustand v5 stores. workflowStore holds the node and edge graph for the active workflow and the list of all saved workflows. executionStore tracks execution status, per-node states, logs and approval state for Human Approval nodes. uiStore manages which panels are open and which node is selected. settingsStore handles user preferences and the encrypted API keys.

The **Workflow Engine Layer** is the execution engine. It validates the graph, builds adjacency lists, runs topological sort via Kahn's algorithm and executes nodes in order using BFS traversal. Each node type has a registered async executor function that takes configuration and input data and returns output data that gets passed downstream.

The **AI Integration Layer** is a unified interface over four AI providers: Groq, Gemini, OpenAI and Anthropic. It handles the different SDKs, response schemas, error formats and streaming APIs behind a single normalized interface. It also contains the HCPT system for context management and the prompt builder for the AI copilot.

The **Storage Layer** uses browser localStorage with Zod validated JSON serialization for workflow data and AES-GCM encrypted storage for API keys.

Fig. 1. System Architecture

C. Workflow Data Format

Workflows are stored as plain JSON. The top level structure looks like this:

CORE ALGORITHMS

A. Execution Engine

Execution happens in four steps. First, graph validation checks that there is at least one trigger or start node, that all edge endpoints reference real nodes, and that all node types have registered executor functions. Second, we build an adjacency list from the edge data. Third, Kahn's algorithm produces a topological ordering. Fourth, BFS traversal exe-cutes nodes in that order passing each node's output to its downstream neighbors as their input.

{
"id": "wf_abc123", "name": "My Workflow", "version": "3.0",
"nodes": [...],
"edges": [...],
"viewport": {"x":0,"y":0,"zoom":1}, "createdAt": "2025-01-01T00:00:00Z", "updatedAt": "2025-01-01T12:00:00Z"
}

Each node has an id, type, position on the canvas and a data object with its configuration. Edges reference source and target node IDs. The whole thing is readable by a human, can be version controlled in Git, shared as a file or imported into any other NODOMO instance without any conversion step.

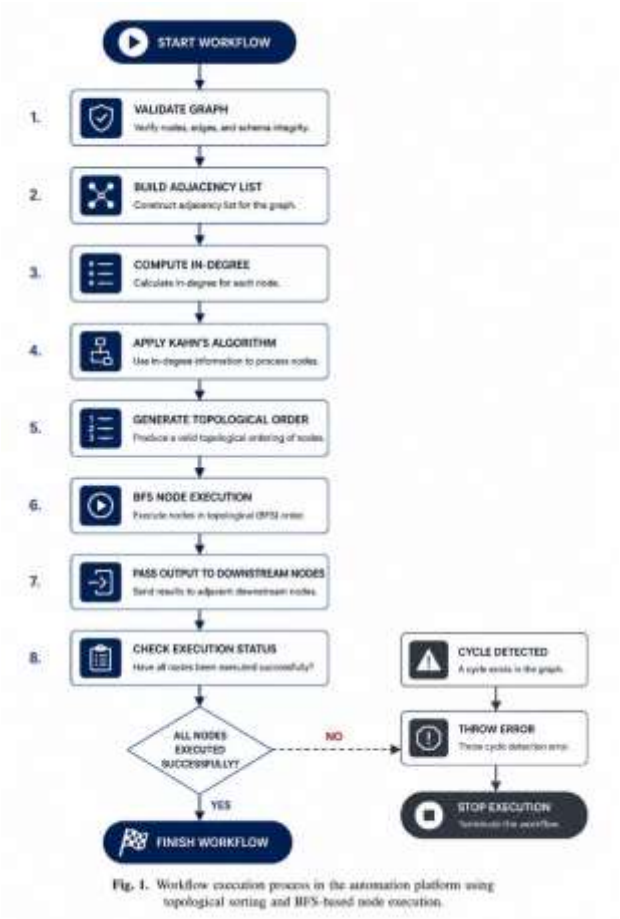

Fig. 2. Workflow Execution Engine flowchart System Architecture

Kahn's algorithm computes in degree for every node. Starts a queue with all zero in degree nodes and then iteratively dequeues and processes nodes while decrementing the in degree of their neighbors:
function topologicalSort(nodes, edges): inDegree = {}
adjacency = {}
for each node in nodes: inDegree[node.id] = 0 adjacency[node.id] = []
for each edge in edges: adjacency[edge.source].push(edge.target)

inDegree[edge.target] += 1 queue = nodes where inDegree == 0 order = []
while queue not empty:

Stage 2 — Template similarity. If no rule fires we compute Jaccard similarity between the current workflow's node type set and the node type sets of 12 built in templates:

node = queue.dequeue() order.push(node)

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

for neighbor in adjacency[node]: inDegree[neighbor] -= 1
if inDegree[neighbor] == 0: queue.enqueue(neighbor)
if len(order) != len(nodes): throw CycleDetectedError
return order

Time complexity is O(V + E) where V is node count and E is edge count. For the workflow sizes NODOMO handles, this is essentially instant.

B. Cycle Detection

We check for cycles at edge creation time rather than at execution time. This way a cycle never makes it into the workflow data model in the first place. When a user draws an edge from a source to a target we temporarily add the proposed edge and run DFS from the source node. If we reach the source node again a cycle exists, the edge is rejected and the user sees a notification explaining why. The DFS uses a recursion stack to track which nodes are currently being visited [14].

C. Node Recommendation Engine

When a user requests a suggestion the recommendation engine goes through three stages and stops at whichever stage first produces a confident answer.

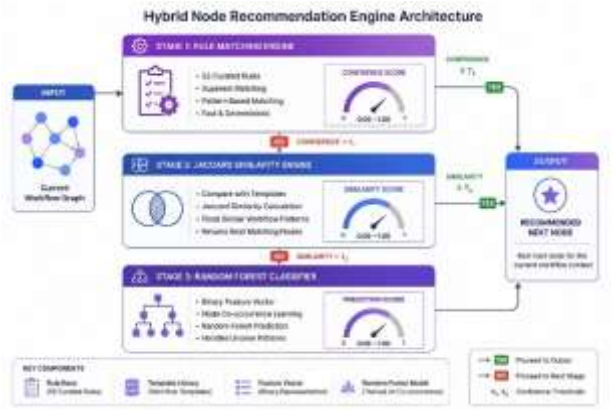

Fig. 3. Hybrid Node Recommendation Engine Architecture

Stage 1 — Rule matching. We have 55 curated rules that map node type combinations to recommended next nodes. A rule says something like : if the current workflow contains {Start, HTTP Request, JSON Parser}, suggest an AI Node.

Rules are evaluated by checking if the current workflow's node type set is a superset of the rule's left-hand side. When multiple rules match the most specific one (with the most conditions) wins.

If the best match scores above 0.3 we recommend nodes from that template that the user has not added yet in template order.

Stage 3 — Random Forest. If no template clears the threshold a Random Forest classifier [12] trained on node co-occurrence patterns gives a probability distribution over possible next node types. The workflow is encoded as a binary feature vector each dimension is whether a particular node type is present. The classifier returns the highest probability node type not already in the workflow.

D. Hierarchical Context Parsing Tree

When you chain multiple LLM calls in a workflow you quickly run into the context window problem. Passing the full output of every upstream node into each LLM call is not practical for long workflows. The Hierarchical Context Parsing Tree (HCPT) solves this by organizing context as a tree that mirrors the workflow DAG.

At each AI node instead of concatenating all ancestor outputs the HCPT traverses the tree and extracts only the context directly relevant to that node. Sub-graph outputs are compressed hierarchically before being passed to the next level. This keeps token usage bounded regardless of how deep the workflow is.

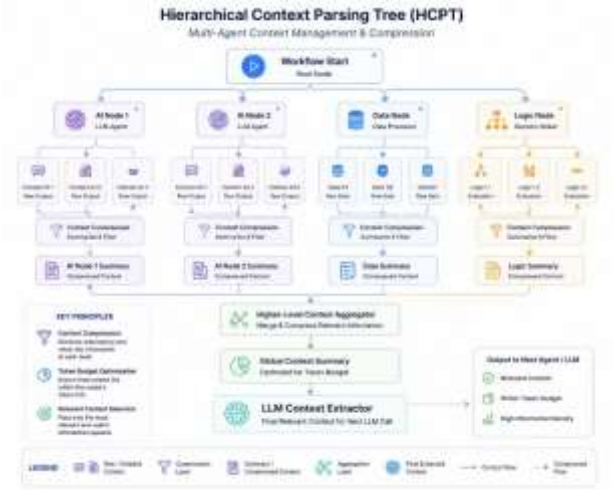

Fig. 4. Hierarchical Context parsing Tree

Formally, for a node n with ancestors A(n), the context provided to n's LLM call is:

$$C(n) = \text{HCPT-Extract}(A(n), \text{budget}(n)) \quad (2)$$

where budget(n) is the token budget for context at node n, set as a fraction of the model's maximum context window.

E. API Key Encryption

Storing API keys in plain text in local Storage is not acceptable because any JavaScript running at the same origin could read them. We encrypt all API keys using AES-GCM 256-bit via the browser's native Web Crypto API [16] [17].

On first launch, the Web Crypto API generates a random 256-bit encryption key stored separately in local Storage. When the user enters an API key it is encrypted with a fresh random 96-bit initialization vector for each operation. The cipher text and IV are stored together. When a node needs to make an API call the key, is decrypted in memory, used immediately and the plain text is discarded. It is never put into React state or passed between components.

GCM mode gives us both confidentiality and integrity. If anything tampers with the stored ciphertext, decryption fails with an authentication error rather than silently returning garbage data [17].

TECHNOLOGY STACK

A. Framework

We used Next.js 15 with the App Router. The landing page is statically generated for performance. The workspace is client side rendered since the interactive canvas does not need server side data fetching. React 19 comes with Next.js 15 and the improved hydration and concurrent rendering features helped with canvas performance.

TypeScript strict mode is enforced throughout the codebase. This is not optional for a project like this. The workflow data model, node schemas, execution engine and stores all depend on type correctness. We use Zod v3 for runtime schema validation on workflow import and on node execution incoming data is validated against the schema before any processing happens.

B. Canvas

The visual canvas is built on @xyflow/react, which is React Flow v12. We chose React Flow over JsPlumb and JointJS because it is React native rather than a wrapper around a framework agnostic library, it has the most active community of any React node editor library and it is used in production in similar applications at companies like Stripe and Typeform. The MIT license works for an open source project.

For layout of AI-generated workflows, we use Dagre. When the copilot generates nodes and edges from a prompt there is no position data. Dagre computes a clean hierarchical layout that gets rendered immediately on the canvas.

C. State Management

Four Zustand v5 stores handle all state. We picked Zustand over Redux because there is no Provider setup, no action/re-ducer boilerplate and no selector optimization needed at this scale. The stores are separated by domain : workflow data, UI state, execution state and settings. TanStack Query v5 handles async server state during execution caching, background refetching, deduplication.

D. AI Providers

We support Groq, Gemini, OpenAI and Anthropic. Each has its own SDK with different response schemas, error formats and streaming implementations. We built a unified abstraction layer that normalizes these into a single generateText(provider, model, systemPrompt, userPrompt, options) interface. Nodes choose their provider independently so a single workflow can use Groq for speed sensitive steps and GPT-4o for steps where quality matters more.

E. Search

The command palette uses Fuse.js for fuzzy search which implements the Bitap algorithm [15] with a tolerance of 0.4. Search covers workflow names, template names, node types and actions. Results are grouped by type and ranked by score.

NODE LIBRARY

A. Design and Organization

NODOMO has over 50 node types organized into seven categories: Trigger (green), AI (purple), Data (blue), Logic (amber), Automation (teal), Content (pink), and Utility/Output (gray). Every node has the same visual anatomy a color stripe on the left border identifying the category, a Lucide React icon, an inline editable label, a status badge showing idle/running/success/error/retrying/paused, input and output connection handles and a millisecond execution time display after the node runs.

B. Node Types

AI nodes cover Groq (Llama 3.3 70B, Mixtral 8x7B, DeepSeek R1 70B), Gemini (1.5 Pro, 2.0 Flash), OpenAI (GPT-4o, GPT-4 Turbo), and Anthropic (Claude 3.5 Sonnet, Claude 3 Haiku). Specialized variants include Summarizer, Classifier, Sentiment Analyzer, AI Planner and AI Validator.

Logic nodes include Condition (JavaScript expression eval-uation with true/false branches), Switch, Loop, Filter, Router (fan-out to parallel paths), Merge (wait-for-all synchroniza-tion), Delay, Retry with configurable backoff, Human Ap-proval and Rate Limiter. The Code node runs inline JavaScript or Python (Python via the Judge0 CE API) for arbitrary custom logic.

Automation nodes handle WhatsApp Business API, Email via SMTP and SendGrid, SMS via Twilio, Slack, Discord, Telegram, and browser push notifications.

Content nodes cover a full content pipeline: Script Gen-erator, Thumbnail Generator via DALL-E and Stability AI, YouTube Upload via Data API v3, Caption Generator, Social Media Publisher, Blog Writer, Image Generator, Voice Gener-ator via ElevenLabs and OpenAI TTS and Transcript Reader via Whisper.

C. Human Approval Node

We want to briefly explain the Human Approval node be-cause it reflects something we feel strongly about. Automation should assist human judgment at critical moments not bypass it entirely. When a workflow hits a Human Approval node execution stops. A modal appears with a custom message, the current data formatted for review and Approve and Cancel buttons. On approval, execution continues from the next node. On cancel or timeout, the workflow ends with a log entry.

This is useful for reviewing AI-generated content before it goes public, confirming high value actions before they execute or validating lists before sending bulk messages. It gives the user a checkpoint without breaking the overall automation flow.

USE CASES

A. Content Creator

A YouTuber with around 50,000 subscribers was spending roughly four hours a week on repetitive content tasks writing scripts, generating thumbnails, pushing descriptions to social platforms, sending newsletter emails. With NODOMO, that same process takes about ten supervised minutes. The work-flow is: Start (topic input) → Script Generator → Human Approval → Thumbnail Generator → YouTube Upload → Caption Generator → Social Media Publisher → Email. The Human Approval step is the only point where the creator needs to actually look at anything before it goes out.

B. Academic Research

A researcher compiling a literature review on some topic can set up a workflow that fetches from multiple source APIs, loops through each result, passes each article through a Groq summarizer, merges the summaries and feeds them to Gemini for synthesis into a formatted report. A few minutes of automated work versus several hours of reading and note taking manually.

C. Small Business

A small e-commerce shop owner who was manually re-sponding to WhatsApp customer messages can use NODOMO to set up: WhatsApp Trigger → AI Classifier → Switch (order inquiry / complaint / general) → appropriate response template via WhatsApp → log to Google Sheets. Customers get immediate responses and the owner has a full log without
spending time on individual replies.

D. Developer Prototyping

A developer who wants to test a multi-step AI pipeline before writing production code can build the whole thing in NODOMO, test it with real APIs and real data, see the output at every node and validate the approach in a few hours instead of writing boilerplate code for a week.

E. Lead Processing

A marketing team receiving leads from a web form can set up: Webhook Trigger → HTTP Request to enrich company data → JSON Parser → Claude AI for lead scoring → Condition on score threshold → personalized Email for high-score leads → Slack notification for the sales team → HTTP Request to update the CRM. The whole thing runs in under
five seconds per lead with no human in the loop.

Security and Privacy

A. No Data Collection

NODOMO collects nothing. No telemetry, no analytics, no usage tracking, nothing is sent to a server because there is no server. This is not a policy we chose — it is a consequence of the architecture. There is nowhere to send data to.

B. API Key Protection

When a user enters an API key, it is encrypted immediately using AES-GCM with a fresh 96-bit IV. The ciphertext and IV are stored in localStorage. The encryption key itself is a random 256-bit key also in local Storage generated on first launch. When a node needs to make an API call the key is decrypted in memory, used, and discarded. The plain text is never stored in component state or passed as a prop.

Fig. 5. API Key Encryption Workflow

If anything modifies the stored ciphertext, GCM's authen-tication tag catches it and decryption throws an error rather than returning corrupted data [17]. Stealing the encrypted keys requires physical access to the device, which is outside the scope of what a browser application needs to defend against.

C. Code Node Sandboxing

The Code node lets users run arbitrary JavaScript or Python. JavaScript runs in a sandboxed context with access to browser globals blocked no document, no window.location, no fetch for non workflow calls. Only the input variable and standard built-ins are available. Python goes through the Judge0 CE API, which handles sandboxed server side execution.

D. Auditability

The entire codebase is on GitHub and open source. The encryption code, the AI integration, the execution engine everything can be read and audited independently. There are no hidden components. If someone finds a vulnerability they can report it publicly and we can fix it in the open.

FUTURE WORK

A. Workflow Sharing Marketplace

The most common thing people asked for during early testing was a way to share workflows as templates. The plan is a searchable gallery where users can browse, preview with a live non interactive canvas and one click import workflows shared by the community. Storage would be on GitHub Gists or a minimal backend, keeping the zero infrastructure approach where possible.

B. Collaborative Editing

We want to add real-time multi-user canvas editing. The approach we are evaluating is CRDTs for conflict-free con-current edits, with cursor presence indicators showing where each collaborator is on the canvas. This would let teams design workflows together without the back-and-forth of version conflicts.

C. Better ML Recommendations

The current Random Forest model is trained on a manually curated dataset of workflow patterns. A better version would use opt-in anonymized usage data from real workflows to retrain the model on actual co-occurrence patterns. This would improve recommendation quality significantly over time as the user base grows.

D. Offline PWA

Making NODOMO a Progressive Web App with service worker caching would let it work fully offline for workflows that use local data or cached API responses. This is espe-cially relevant for enterprise self-hosting where network access might be restricted.

CONCLUSION

NODOMO started from a frustration that most of us in the team had personally experienced: workflow automation is theoretically accessible but practically limited to people who already know how to code. We wanted to build something that actually closed that gap rather than just narrowing it slightly. The core technical things we are most happy with are the browser native architecture with no server dependency the three-stage recommendation engine that handles common patterns quickly and unusual ones with ML, the HCPT system that makes multi-LLM workflows practical without context window problems, and the AES-GCM encryption that keeps API keys safe without any backend involvement.

But the more important part is the philosophy behind the design. Automation tools should not need a Dev-Ops engineer to set up. User data should not have to leave the user's device. The tool should be free and open and readable and AI should be built into the core of the system not attached to the side of something that was designed without it.

Whether NODOMO actually succeeds in making automa-tion more accessible is something users will

determine. But we think the combination of zero server architecture, AI-native design, a practical node library and full open-source transparency gives it a real shot at being meaningfully different from what currently exists. We are still actively working on it and will continue to improve it based on what people actually need.

REFERENCES

1. McKinsey & Company, "The State of AI: Global Survey," McK-insey Digital, 2024. [Online]. Available: https://www.mckinsey.com/ capabilities/quantumblack/our-insights/the-state-of-ai
2. Z. Li, S. Xu, K. Mei, W. Hua et al., "AutoFlow: Automated Work-flow Generation for Large Language Model Agents," arXiv preprint arXiv:2407.12821, 2024.
3. S. Fan, X. Cong, Z. Fu et al., "WorkflowLLM: Enhancing Workflow Orchestration Capability of Large Language Models," arXiv preprint arXiv:2411.05451, 2024.
4. L. Zhang et al., "AFLOW: Automating Agentic Workflow Generation," in Proc. ICLR 2025, 2025.
5. M. Kong, Z. Qu, Z. Zhou et al., "Workflow-R1: Group Sub-Sequence Policy Optimization for Multi-Turn Workflow Construction," arXiv preprint arXiv:2602.01202, 2026.
6. O. Khattab, A. Singhvi, P. Maheshwari et al., "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines," arXiv preprint arXiv:2310.03714, 2023.
7. Q. Wu, G. Bansal, J. Zhang et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," arXiv preprint arXiv:2308.08155, 2023.
8. J. Wei, X. Wang, D. Schuurmans et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in Proc. NeurIPS 2022, 2022.
9. M. Li, Y. Zhao, B. Yu et al., "API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs," in Proc. EMNLP 2023, 2023.
10. M. Besta, N. Blach, A. Kubicek et al., "Graph of Thoughts: Solving Elaborate Problems with Large Language Models," in Proc. AAAI 2024, 2024.
11. Y. Cai, S. Mao, W. Wu et al., "Low-code LLM: Graphical User Interface over Large Language Models," arXiv preprint arXiv:2304.08103, 2024.
12. L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
13. D. E. Knuth, The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd ed. Addison-Wesley, 1997.
14. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 4th ed. MIT Press, 2022.
15. H. S. Baird and H. Goyal, "Improved Approximate String Matching," 1987. (Basis for Bitap/Shift-Or algorithm.)
16. NIST, "Advanced Encryption Standard (AES)," FIPS Publication 197, 2001.
17. D. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM)," NIST Submission, 2004.
18. A. Vaswani, N. Shazeer, N. Parmar et al., "Attention Is All You Need," in Proc. NeurIPS 2017, 2017.
19. Capgemini Research Institute, "Rise of Agentic AI: How Trust is the Key to Human-AI Collaboration," Capgemini, 2025.