

NEXUS: A Visual Low-Code Platform for Designing, Executing, and Deploying AI Automation Pipelines

Usha N¹

Assistant Professor

Faculty of Computing & IT

GM University

Davangere, Karnataka, India-
577006Usha.narayanshetty12@gmail.com
[m](mailto:Usha.narayanshetty12@gmail.com)**Guneshwari Hiremath²**

II-Year MCA

Faculty of Computing & IT

GM University

Davangere, Karnataka, India-
577006guneshwarihiremath@gmail.com**Srujan M³**

II-Year MCA

Faculty of Computing & IT

GM University

Davangere, Karnataka, India-
577006srujanmahadev437@gmail.com

Abstract

Existing AI workflow and low-code automation platforms are designed primarily for skilled developers and enterprise contexts, which makes it challenging for beginners to understand the workings of internal workflow. In this context, this paper presents NEXUS – a low-code platform which is designed to enable the construction, execution, and observation of AI automation pipelines using a node-based graphical interface. Workflows are implemented as directed acyclic graphs (DAGs) composed of the nodes, such as trigger, input, processing, logic, AI, API, and output. The engine for the execution of the workflow handles the flow of data and visually highlights the active nodes and the connections in real time. In addition, NEXUS has a validation procedure which allows it to identify any cyclic dependencies beforehand before the workflow execution. The paper outlines system architecture, workflow representation, DAG validation, execution schedule, and framework for evaluation of execution scalability, validation effectiveness, complexity of the workflow, and bottlenecks of nodes. Since there are no benchmarking practices so far, performance results are unavailable. The main achievement of the paper is the introduction of a user-friendly platform which pays attention to transparency and offers reproducible framework for systematic empirical investigation.

Keywords — Artificial Intelligence, Low-Code Development, Workflow Automation, Visual Programming, AI Pipelines, Intelligent Agents, Directed Acyclic Graph, Workflow Orchestration.

I. Introduction

The advent of Artificial Intelligence (AI) as well as automation technologies has greatly influenced the evolution of contemporary computing ecosystems, which allows the products to accomplish sophisticated actions such as data processing, decision-making, and smart human-computer interaction. The automation pipelines, which are driven by AI, have become a necessary element of businesses, namely automating processes and utilizing intelligent helpers as well as large-scale systems for data analysis. Although different systems for automation and orchestration of workflows have already appeared [1]-[5], most of those solutions are primarily meant for software engineers working in the field and are originally engineered for use in the corporate environment, making it extremely complicated for a novice user to understand how they operate [6]-[8].

A. Problem Statement

The process of moving from being a consumer of AI tools to being a developer of AI automation pipelines is impeded for novice users and students alike by two factors: first, developing a pipeline normally requires programming knowledge and knowledge of backend configuration; second, even when tools for visual development are available, the workings of the resulting pipeline are often hidden from view, meaning that users do not know how data is flowing through the system and how branching decisions are made. This paper presents a solution to both problems, addressing them simultaneously within the same solution.

B. Research Gap

The literature review presented in Table I indicates that existing visual/no-code AI workflow solutions either (i) facilitate workflow design while missing step-by-step explanations for novices [1], (ii) ease understanding through LLM-generated tips but lack visibility into the underlying execution [2], (iii) intentionally hide reasoning across agents from end users [3], (iv) are controlled by agents rather than by users [4], or (v) deal with advanced MLOps matters instead of educational accessibility [5]. Research about intelligent low-code solutions [6] has not resulted in an actual proven approach and studies to combine low-code with conversational AI [7], that aim for democratizing AI development [8], that involve human-guided no-code AutoML [9], and that explore AI-assisted programming by end users [10] have similar topics in focus but still provide no node-based visual execution-enabled platform suitable for beginners.

C. Research Questions

This document delineates the subsequent research questions that govern the assessing framework in Section VI:

RQ1: Is the scalability of execution time different for various pipeline complexity types (linear, branched, API-based, AI-based, or mixed)?

RQ2: Is the end-to-end execution latency proportional to the memory requirements as the workflow amount (number of nodes) increases?

RQ3: Does the DAG validation process efficiently and successfully identify invalid (cyclic) workflows regardless of the size of the graph being used?

RQ4: Are there any node types that greatly contribute to the workflow time taken to execute?

D. Contributions

The primary contributions of the present research are: (1) the development of NEXUS, a node-based visual low-code platform to create, execute, and visualize AI automation pipelines aimed at novice users; (2) a formal DAG-based validation which serves to identify structurally incorrect workflows before the actual execution; (3) a real-time visualization mechanism that operates at the node/edge level and shows how the internal pipeline operates; (4) a framework for evaluating the system presented in the form of four research questions with specified dependent and independent variables and indicators allowing for an empirical assessment of the performance in execution and validation of the system. Statements about possible improvement in the learning process or efficiency of construction of the workflows are treated in this work as design goals behind the implementation of (1)-(3).

II. Literature Survey

Sl. No	Paper Title	Author(s) & Year	Objective of Study	Methodology Used	Key Findings / Results	Research Gap Identified
1	Rapsai: Accelerating Machine Learning Prototyping of Multimedia Applications Through Visual Programming	R. Du et al. (2023)	To enable rapid, code-free prototyping of end-to-end multimedia ML pipelines through a visual node-graph editor.	Node-graph visual programming platform (Visual Blocks for ML) with drag-and-drop input, processing, model, and visualization nodes; real-time in-browser execution.	Significantly accelerated ML pipeline prototyping; enabled interactive model evaluation, data augmentation, and result comparison for non-experts.	Lacks structured pedagogical, step-by-step execution visualization and support for general automation logic (triggers, branching, APIs) for beginners.
2	InstructPipe: Generating Visual Blocks Pipelines with Human Instructions and LLMs	Z. Zhou et al. (2025)	To reduce the learning curve of visual ML pipeline construction using LLM-generated	LLM-based assistant (two LLM modules + code interpreter) integrated into the	User study (N=16) showed reduced workload and learning curve; enabled open-	Prioritizes automated pipeline generation over transparent, real-time execution

Sl. No	Paper Title	Author(s) & Year	Objective of Study	Methodology Used	Key Findings / Results	Research Gap Identified
			suggestions from natural-language instructions.	Visual Blocks node-graph editor.	ended pipeline exploration.	visualization needed to teach internal pipeline mechanics.
3	AIAP: A No-Code Workflow Builder for Non-Experts with Natural Language and Multi-Agent Collaboration	H. An et al. (2025)	To help non-experts build AI services by combining natural-language input with visual workflow construction.	No-code platform with a coordinated multi-agent system decomposing ambiguous instructions into modular workflow steps.	User study (N=32) showed improved intuitive service development through AI suggestions and modular workflows.	Hides internal multi-agent reasoning from users, limiting transparency and understanding of pipeline internals.
4	ProAgent: From Robotic Process Automation to Agentic Process Automation	Y. Ye et al. (2023)	To evolve Robotic Process Automation into Agentic Process Automation using LLM-based agents for workflow construction and execution.	Agentic Workflow Description Language separating data-flow and control-flow; LLM agents construct and execute workflow graphs.	Improved workflow construction flexibility and execution survival rate over static RPA and ReAct baselines.	No beginner-facing visual editor or real-time execution visualization; workflows are agent-authored rather than learner-controlled.
5	Data Analytics Environment: Combining Visual Programming and MLOps for AI Workflow Creation	A. Grilo et al. (2024)	To combine visual programming with MLOps practices for creating and operationalizing AI workflows.	Node-based visual environment integrated with MLOps tools for versioning, monitoring, and deployment.	Bridged exploratory visual pipeline design with production-grade MLOps concerns such as reproducibility and deployment.	Enterprise/MLOps complexity is unsuitable for beginner education; lacks pedagogical execution visualization.
6	Approach to the Implementation of Intelligent Low-Code Platforms	Y. I. Rogozov et al. (2022)	To propose an architecture for embedding intelligent capabilities into low-code platforms.	Conceptual architectural design covering component reuse and semi-automated workflow assembly.	Provided a generalizable architectural pattern for intelligent low-code platforms.	No concrete implementation, usability study, or real-time execution visualization.
7	Combining Low-Code Development with ChatGPT to Novel No-Code Approaches: A	J. Martins et al. (2023)	To explore how integrating ChatGPT into low-code development can evolve it toward	Qualitative focus-group study examining the low-code software development life cycle with ChatGPT.	Proposed an extended low-code life-cycle model incorporating human-and-	No working visual pipeline editor or execution engine was implemented or evaluated.

Sl. No	Paper Title	Author(s) & Year	Objective of Study	Methodology Used	Key Findings / Results	Research Gap Identified
	Focus-Group Study		no-code approaches.		ChatGPT collaboration.	
8	Democratizing Artificial Intelligence: How No-Code AI Can Leverage Machine Learning Operations	L. Sundberg and J. Holmström (2023)	To examine how no-code AI intersects with MLOps to democratize AI development for non-specialists.	Conceptual and case-based analysis of no-code AI adoption within organizations.	Highlighted the democratization value and governance considerations of no-code AI tools.	Business/management perspective only; no technical or educational tooling proposed.
9	Human-Centered AI Product Prototyping with No-Code AutoML: Conceptual Framework, Potentials and Limitations	M. Truss and M. Schmitt (2024)	To develop a human-centered framework for prototyping AI products using no-code AutoML tools.	Conceptual framework evaluated against usability and design-thinking criteria.	Identified design principles for making no-code AutoML tools more human-centered and accessible.	Scoped to AutoML model-building only; no broader automation-pipeline (trigger/logic/API) support.
10	Feasibility of AI-Assisted Programming for End-User Development	I. Weber (2025)	To assess the feasibility of AI-assisted natural-language programming as an alternative to visual LCNC platforms.	Feasibility/comparative analysis of AI-assisted natural-language coding versus established visual LCNC approaches.	Highlighted trade-offs in flexibility, learning curve, and vendor lock-in between the two paradigms.	No visual, node-based, execution-aware learning platform proposed for end-user development.

TABLE I. Comparative Literature Survey on Visual and No-Code AI Workflow Platforms

III. Related Work

A. Visual Programming

Visual programming allows users to develop workflows via graphical user interfaces instead of using code, modeling computational processes as interconnected nodes on a layout. Such technologies have been used in a variety of applications, ranging from data science to the building of machine learning processes to automating business processes. They reduce complexity of the developing process, making it easier for non-coders to use the programming solutions[1].

B. Low-Code/No-Code AI

Low-code and no-code development enables the creation of applications with little coding knowledge through the provision of tools such as reusable modules, graphical editors, and user-friendly settings configurations [7], [8]. In enterprises, low-code/no-code technologies are already gaining popularity for quick application development [8], while human-centered creation ideas have been introduced to increase the accessibility of such tools for people without technical background [9].

C. AI Workflow Orchestration

Workflow orchestration refers to the systems that are engaged in coordinating various tasks such as preparing data, developing models as well as execution and deployment of services [5]. The approach that includes the possibility of creating workflows utilizing LLM-based agents in order to develop the workflow graphs straight from various natural language instructions is called agentic methods [3], [4]. However, these systems are mainly designed for people with extensive experience in programming in order to develop workflows, or they put creating of workflows on the agent [4], [5], [10].

D. Educational Workflow Systems

A few researchers have focused more specifically on accessibility issues by looking into the use of LLM-generated pieces of advice for making the process of learning visual pipeline construction easier [2] and on how low-code development processes change when coupled with conversational AI support [7]. Even though those systems reduce construction work, they do not pay enough attention to how visualizing the construction process in real time could be used to teach pipeline construction methods.

E. Positioning of This Work

Contrary to systems that focus on automated pipeline production [2], keep multi-agent reasoning secret from users [3], or make agents produce rather than learners manage pipelines [4] systems, NEXUS enables the user to create their own workflow by keeping the learner in full control of engine execution functionalities like when nodes are activated and how data flows and branches are evaluated. This makes it possible for NEXUS to work alongside the systems presented in Table I, such as MLOps systems [5] or enterprise systems [6] and [8], which focus on production systems.

IV. NEXUS System Design

A. Design Goals

NEXUS has four principles guiding its design, which are (1) Accessibility — creating a pipeline should not involve scripting; (2) Transparency — execution should be discerned actively and not just as a result; (3) Reliability — invalid workflows must be easily identified and disallowed before execution; and (4) Modularity — adding new types of nodes must not call for alterations to NEXUS' core.

B. System Architecture

The NEXUS framework has six distinct layers which are illustrated in Fig. 1. The first layer is the Presentation Layer, which includes a drag-and-drop visual editor. The second is the Workflow Management Layer that processes workflow information while the Execution Engine is responsible for the execution and scheduling of nodes. The Node Registry serves as a repository of the different types of nodes available for use. The Integration Layer lets you connect with third-party AI services and lastly, the Persistence Layer saves workflows.

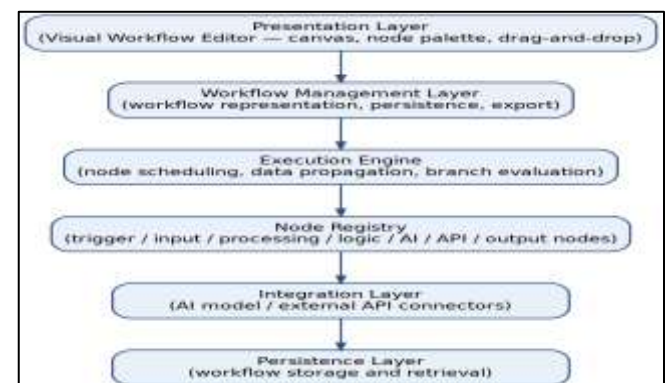


Fig. 1. Layered architecture of the NEXUS platform.

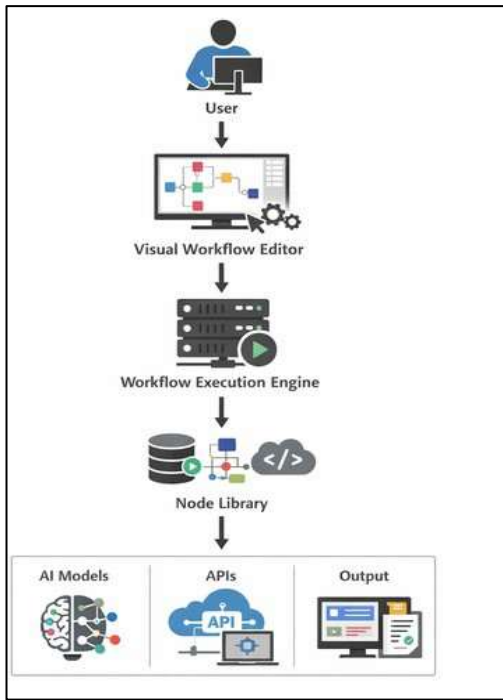


Fig. 2. Conceptual overview of the NEXUS platform, from user interaction through the visual editor and execution engine to AI models, APIs, and output.

C. Workflow Representation

A workflow is modelled as a directed graph $G = (V, E)$, where every node $v \in V$ stands for a functional unit (trigger, input, processing, logic, AI, API, or output) and where each directed edge $e \in E$ represents the data propagation from the output of one node to the input of another. Every node has an identifier, type, configuration object corresponding to its type, a number of input ports, a number of output ports, and an execution state (idle, running, finished or failed). The workflow must be free of cycles; hence, G must be a DAG and must contain at least one trigger node with no incoming edges.

D. Node Model

Table II lists the node categories currently supported by the NEXUS node library and their functional role within a pipeline.

Node Category	Function	Example Role in a Pipeline
Trigger Node	Initiates workflow execution manually or via an external event	Starts the pipeline when a user runs the workflow
Input Node	Introduces user-provided or external data into the workflow	Supplies raw text, files, or form data to the pipeline

Node Category	Function	Example Role in a Pipeline
Processing Node	Performs transformations such as text formatting, data manipulation, or template generation	Reformats or restructures data between pipeline stages
Logic Node	Evaluates conditions and determines branching direction	Routes execution along different paths based on a condition
AI Node	Interacts with language models or intelligent agents	Generates text, classifications, or analysis via an AI model/API
API Node	Communicates with external services over HTTP-based interfaces	Sends/receives data to a third-party API
Output Node	Displays or returns the final results produced by the pipeline	Presents the final pipeline result to the user

TABLE II. NEXUS Node Categories and Functionality

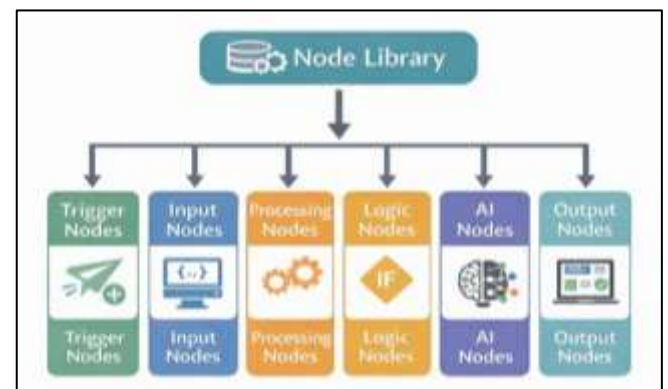


Fig. 3. Categories of nodes available in the NEXUS workflow platform.

E. Execution Engine

During the execution of workflows, the first step carried out by the engine is to identify the trigger node and check the integrity of the graph (Section IV-F). The following step is to process different nodes in a manner that is consistent with the dependencies of the graph. At execution time, each of the conditional nodes is evaluated to establish which downstream path will be taken.

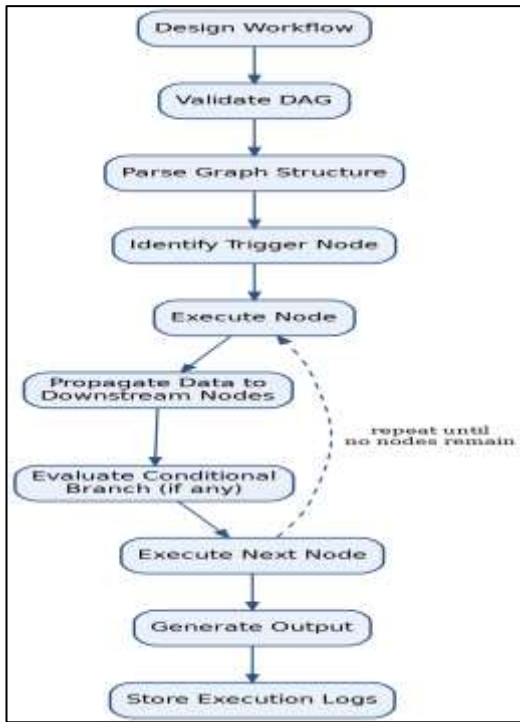


Fig. 4. Workflow execution lifecycle in the NEXUS engine.

Algorithm 2 details the process of scheduling execution in line with our design principles, whereby nodes are processed using the topological approach as per G , in that no one node can be scheduled without its predecessor nodes being executed before it.

Algorithm 2: Execution Scheduling Algorithm

Input: validated DAG $G = (V, E)$, with t being trigger node

Output: execution status and output value of every node

```

1 queue <- [t]
2 for every node v in V, in_degree[v] <- number of
  incoming edges of v
3 until the queue is empty
4   node <- queue.dequeue()
5   input <- gather output from the node's predecessors
6   if node.type == LOGIC
7     branch <- evaluating the node.condition using input
8     active_successors <- successors on a branch
9   else
10    output <- node.process(input)
11    active_successors <- successors of the node
12 for every s in active_successors
13   in_degree[s] <- in_degree[s] - 1
14   if in_degree[s] == 0
15     queue.enqueue(s)
16 return execution status log

```

F. DAG Validation

NEXUS performs a check that the workflow being executed is indeed a DAG (Directed Acyclic Graph) by employing the cycle detection algorithm based on depth-first search (refer to Algorithm 1). The workflow presented in Fig. 5 demonstrates an example of both a valid workflow and an invalid workflow which contains a cycle.

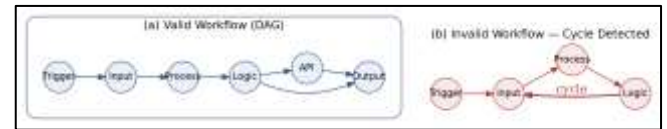


Fig. 5. (a) A valid NEXUS workflow forming a DAG. (b) An invalid workflow containing a cycle, which validation rejects prior to execution.

Algorithm 1: Finding Cycles in a Directed Acyclic Graph

Input: Any graph $G = (V, E)$

Output: Return either valid (true) or invalid with identification of the cycle (false)

```

1 Set color[v] to WHITE,  $\forall v$  in V
2 for each v in V:
3   If color[v] = WHITE:
4     if DFS_VISIT(v) = false then return false
5 Return true

```

DFS_VISIT(v):

```

6 Set color[v] to GRAY
7 for each successor s of v:
8   If color[s] = GRAY return false
9   If color[s] = WHITE and DFS_VISIT(s) = false
    return false
10 Set color[v] to BLACK
11 Return true

```

This is an algorithm based on the depth-first search principle and gives correct results under the assumption that it visits all the edges of the graph. Its time and space complexity is $O(|V| + |E|)$ and $O(|V|)$ respectively. The analysis further in Section VI will be concentrated on evaluation of the runtime and efficiency rather than confirming correctness of calculations.

V. Implementation

A. Visual Workflow Editor

The application has a graphic workflow editor that allows the users to design automation workflows in a drag-and-drop fashion. The user takes nodes from the library of

nodes and places them on a canvas. The nodes are linked with directed links that indicate the flow of information and the execution order. It is possible for the user to move the nodes, change their properties, and see how the pipeline is built before it is run.

B. Node Library

The library of nodes (Table II) contains reusable functional components that include triggers, data inputs, data processing functions, logic, AI models, APIs, and output display components. The modular architecture of the library makes it possible to add new types of nodes without any modifications in the base execution engine and meet the modularity design requirements listed in Section IV-A.

C. Workflow Execution

The execution engine (Section IV-E, Algorithm 2) is responsible for transferring data between the nodes during execution. Each node receives its input, processes it according to the defined algorithm, and sends the results to its next nodes. The visual interface highlights the active nodes on the canvas during the execution process.

D. AI/API Integration

AI nodes work alongside other language models or intelligent agents in order to produce responses or carry out data examinations, while API nodes communicate with outsiders via HTTP interfaces. As these processes depend on outside services that are accessible over the network, it follows that the execution engine of NEXUS cannot control their delays or reliability; both of these factors are taken into account in the evaluation framework laid out in Section VI, as well as in the limitations discussed in Section VII.

E. Persistence and Export

The ability to save constructed pipelines means that users can retrieve them at a later time for editing or execution, facilitating experimentation with different types of workflows.

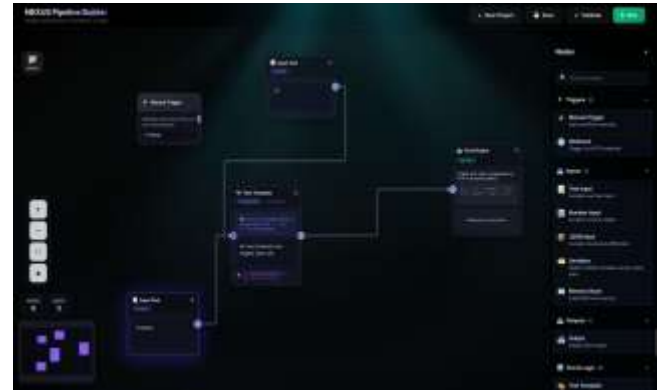


Fig. 6. Example AI automation workflow created using the NEXUS visual workflow editor, combining input, AI, conditional, and output nodes.

VI. Evaluation Framework

This portion outlines the method to evaluate RQ1-RQ4. At the moment of writing, no experiments have been conducted, hence this portion explains the methodology, metrics, and variables allowing the evaluation to be replicable once benchmark data are available. No performance statistics, graphs, or results will be presented in this article, which goes along with the principle that claims that were never tested cannot be treated as results.

R Q	Independent Variable	Dependent Variable	Experiment / Data Collection	Planned Statistical Analysis
R Q 1	Pipeline type (linear / branched / API-based / AI-based / mixed)	Execution time, variance	Benchmark five representative pipeline types, 20-30 runs each	One-way comparison of means across types (e.g., ANOVA or Kruskal-Wallis, selected after checking distributional assumptions)
R Q 2	Node count (5, 10, 20, 50, 100, 200)	End-to-end execution time, peak memory	Generate benchmark workflows per size, 20-30 runs each, engine-only latency isolated from external API/AI latency	Regression fit (linear vs. polynomial) of mean latency and memory against node count; report R^2

R Q	Independent Variable	Dependent Variable	Experiment / Data Collection	Planned Statistical Analysis
R Q 3	Graph size and cycle presence/location	Validation time; detection accuracy	Generate valid and intentionally cyclic graphs across the RQ2 size range, 20-30 runs each	Descriptive statistics (accuracy %, mean/95% CI validation time); accuracy expected at 100% given Algorithm 1's deterministic guarantee
R Q 4	Node type	Per-node execution duration	Aggregate node-level logs collected during RQ1-RQ3 runs, grouped by node type	Descriptive comparison of per-type duration distributions (e.g., box plots)

TABLE III. Planned Evaluation Design for RQ1-RQ4

Instrumentation necessary for data gathering consists of an execution logger (which will record per-node start/end timestamps, input/output sizes and status), a performance monitor (which will record per-run peak memory and, where applicable, CPU usage) and a workflow benchmark generator (which will provide programmatic creation of graphs of given size and shape, including cyclic graphs) along with a data exporter writing logs in CSV format for offline statistical analysis. A user study comparing visualization in NEXUS and code-based programming, as well as a pre-/post-test study of educational effectiveness are recognized as useful ones but are not considered to be within the framework of evaluation in this case because of the necessity to recruit participants and to go through ethics and consent procedures according to the institutional policy.

VII. Limitations and Threats to Validity

The library of nodes has only a limited number of categories (Table II), and operations needing something aside from this list are not feasible at this moment. AI and API nodes rely upon other companies so the reality of applying the performance measurements must include the possibility of latency in performing through the “network” processes of connecting to these external

services. The method explained in Section VI discusses how to identify latency pertaining just to the nodes. So far, it should be said that the data on successful operation of this library hasn’t been collected in any large-scale or production evaluations yet. It is said in Section VI that the realization of the improved performance of accessibility, understanding, or construction ability relative to code-based development hasn’t been evidenced yet; that is why all the statements about successful self-fulfillment haven’t been documented yet. Finally, the existing DAG algorithm proves that the structural cycle detection occurs but gives no guarantees about correctness in terms of semantics (e.g., logical command with irrelevant condition) — this issue threatens reliability of the developed approach.

VIII. Conclusion and Future Work

This article introduced NEXUS, which is a low-code solution that utilizes visual components for the creation, use, and observation of AI automation pipelines using a user-friendly node-centered platform. The platform utilizes direct acyclic graphs (DAGs) to structure its procedures, ensure the correctness of the architecture before its usage, and visualize the functions of each node in real time for the purpose of easing the usage of inexperienced and educational users. Parts IV and V of the paper offer detailed information on the architecture of the system, its procedures along with the components of the platform and the methods for control of DAG structures and timings.

The directions for the future work include the implementation of the tools mentioned in Section VI and conducting the benchmarks PQ1-RQ4 for the finalization of the Results chapter of the current paper. Furthermore, future work will involve the implementation of comparisons between visual and coding workflows and pre- and post-tests regarding the education effectiveness.

References

- [1] R. Du, N. Li, J. Jin, M. Carney, S. Miles, M. Kleiner, X. Yuan, Y. Zhang, A. Kulkarni, X. Liu, A. Sabie, S. Orts-Escolano, A. Kar, P. Yu, R. Iyengar, A. Kowdle and A. Olwal, "Rapsai: Accelerating machine learning prototyping of multimedia applications through visual programming," in Proc. of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23), 2023, Article Number 125, doi: 10.1145/3544548.3581338.

- [2] Z. Zhou, J. Jin, V. Phadnis, X. Yuan, J. Jiang, X. Qian, J. Zhou, Y. Huang, Z. Xu, Y. Zhang, K. Wright, J. Mayes, M. Sherwood, J. Lee, A. Olwal, D. Kim, R. Iyengar, N. Li and R. Du, "InstructPipe: Generating visual blocks pipelines with human instructions and LLMs," in Proc. of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25), 2025, doi: 10.1145/3706598.3713905.
- [3] H. An, Y. Kim, W. Seo, J. Park, D. Kang, C. Oh, D. Kim and S. Lee, "AIAP: A no-code workflow builder for non-experts with natural language and multi-agent collaboration," arXiv:2508.02470, 2025.
- [4] Y. Ye, X. Cong, S. Tian, J. Cao, H. Wang, Y. Qin, Y. Lu, H. Yu, H. Wang, Y. Lin, Z. Liu and M. Sun, "ProAgent: From robotic process automation to agentic process automation," arXiv:2311.10751, 2023.
- [5] A. Grilo, P. Figueiras, B. Rêga, L. Lourenço, A. Khodamoradi, R. Costa, and R. Jardim-Gonçalves, "Data analytics environment: Combining visual programming and MLOps for AI workflow creation," in Proc. 2024 IEEE Int. Conf. Eng., Technol., Innov. (ICE/ITMC), 2024, pp. 1-9, doi: 10.1109/ICE/ITMC61926.2024.10794244.
- [6] Y. I. Rogozov, V. S. Lapshin, and M. A. Borovskaya, "Approach to the implementation of intelligent low-code platforms," in Proc. Int. Conf. Intell. Inf. Technol. Ind., Cham: Springer, 2022, pp. 42-50, doi: 10.1007/978-3-031-19620-1_5.
- [7] J. Martins, F. Branco, and H. Mamede, "Combining low-code development with ChatGPT to novel no-code approaches: A focus-group study," Intell. Syst. Appl., vol. 20, Art. no. 200289, 2023.
- [8] L. Sundberg and J. Holmström, "Democratizing artificial intelligence: How no-code AI can leverage machine learning operations," Bus. Horizons, vol. 66, no. 6, pp. 777-788, 2023.
- [9] M. Truss and M. Schmitt, "Human-centered AI product prototyping with no-code AutoML: Conceptual framework, potentials and limitations," Int. J. Human-Comput. Interact., 2024.
- [10] I. Weber, "Feasibility of AI-assisted programming for end-user development," preprint, accepted for NLPiR 2025.