Journal of Artificial Intelligence and Technology, (Ahead of Print)
https://doi.org/10.37965/jait.2026.0982

ISTP
RESEARCH ARTICLE

AI-Driven Auto-Architecting of Scalable Cloud Pipelines Using Workflow Analytics and Meta-Learning

Harish Kumar Krishnamurthy Sukumar,[1] Navneet Dalipkumar Magotra,[2] Shrutika Prakash Mokashi,[3] Surya Narayana Reddy Chintacunta,[4] and Rohit Laheri[5]

[1]Department of Information Science, Shri Dharmasthala Manjunatheshwara College of Engineering and Technology, Dharwad, Karnataka, 580002, India

[2]Department of Computer Science, Binghamton University, Binghamton 13902, New York, USA

[3]Department of Business Intelligence, Northeastern University, New Haven, CT, USA

[4]Department of Data Science, University of Maryland Baltimore County, Jersey City, NJ, USA

[5]Department of Computer Science & Engineering, Lovely Professional University Phagwara, Jalandhar, India

(Received 25 October 2025; Revised 14 March 2026; Accepted 16 April 2026; Published online 16 May 2026)

Abstract: With the rapid expansion of artificial intelligence (AI) across industries, there is a growing demand for intelligent cloud systems capable of adapting to dynamic workloads and scaling autonomously. Traditional cloud pipeline architectures remain static and manually configured, resulting in performance bottlenecks, inefficient resource utilization, and poor scalability. To overcome these limitations, this study proposes an AI-driven cloud pipeline architecture that leverages historical workflow analytics for the autonomous generation and optimization of cloud pipeline architectures. At its core, the proposed AI-driven cloud pipeline architecture introduces adaptive modular intelligence (AMI), which modularizes tasks based on historical patterns through a dynamic Infrastructure-as-Code (IaC) configuration strategy. It further integrates Flexible Batch Processing Scheduling (FBPS) to intelligently switch between batch and stream processing modes based on workload urgency and system load, enabling real-time scheduling flexibility. To enhance optimization, the Evolutionary Particle Genetic Optimizer (EPGO) refines scheduling configurations through a multi-objective fitness function balancing throughput, latency, and resource efficiency. In parallel, meta-orchestration system (MOS) manages encrypted communication between modules adopting symmetric key encryption, ensuring secure data exchange and protection against unauthorized access or tampering. MOS also tracks configuration changes and dynamically adapts pipeline behaviors without manual intervention, providing a robust governance layer for secure execution, real-time monitoring, and version-controlled reproducibility across the entire cloud infrastructure. Experimental results confirm that the proposed framework surpasses existing methods such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Hybrid Genetic Algorithm (HGA), and Hybrid Parallel Enhanced Genetic Algorithm (HPEGA), achieving a minimum MAPE of 4.1%, faster convergence, improved scalability, and higher overall performance in dynamic AI-driven cloud environments.

Keywords: adaptive modular intelligence; cloud pipeline automation; Evolutionary Particle Genetic Optimizer; Flexible Batch Processing Scheduling; meta-orchestration system

I. INTRODUCTION

The rapid creation of artificial intelligence (AI) resulted in major changes within several industries, which has led to an increasing need of AI professionals. The emergence of cloud computing offers significant support to industries, which provides users with quick access to essential resources, the capacity to extend and contract with need, and enhanced efficiency [1,2]. Nevertheless, the evolving needs and numerous options of implementing AI workloads complicate it with conventional cloud designs. In the traditional cases, the resources are usually allocated beforehand, the changes are performed manually, and the whole process is executed in accordance with the predetermined rules [3,4]. It is against this reason that there is an essential requirement to find solutions for how AI will adapt to the changing demands, operate effectively, and remain dependable.

Whereas there has been an enhancement in cloud automation and orchestration, systems employed to control pipelines are largely configured to perform fixed and rigid operations [5–7]. Most methods of cloud pipeline architecture today demand a significant portion of expert help and make the process more complex in software creation, reducing the ability to respond quickly to changes in demand. In addition, static scheduling and resource allocation imply that any unused historical workflow data in the form of how resources were utilized and why past receivables failed cannot be utilized to enhance future executions [8–10]. The performance of AI cloud platforms is unsatisfactory, since it is unable to develop adaptive learning; furthermore, their circuitry requires additional time and energy that negatively affects their efficiency.

Corresponding author: Harish Kumar Krishnamurthy Sukumar (e-mail: Harishks6312@gmail.com).

Machine learning (ML) and optimization approaches have been examined to speed up cloud services; however, merging them into whole autonomous and flexible systems is not easy. Researchers have found that scheduling batch jobs dynamically and tweaking the cloud pipeline architecture are essential yet not well-explored topics [11–13]. Being able to adjust the setup of AI pipelines when tasks change in real time, yet keep everything reproducible with minimal manual operations, is very important for cloud AI [14,15].

Even with these developments, the current cloud pipeline models continue to fail to autonomously create scalable designs that adapt to the ever-changing workload and provide secure orchestration and resource optimization. Most of the existing solutions would solve independent elements like scheduling or monitoring instead of offering a single intelligent pipeline design. These limitations highlight the need for an integrated AI-driven framework capable of automatically constructing, optimizing, and securely managing cloud pipelines in dynamic computing environments:

- The proposed solution includes a novel AI-based workflow auto-architecting cloud pipeline architecture that relies on historical workflow analytics and meta-learning as its core to create scalable and adaptable cloud pipeline architectures, eliminating the need to rely on manual elasticity configuration, and instead of depending on a predefined infrastructure design.
- An adaptive modular intelligence (AMI) system is presented that autonomously builds a modular pipeline structure through behavior-based task analysis and directed acyclic graph (DAG) modeling and generates Infrastructure-as-Code (IaC) dynamically to be deployed efficiently across diverse clouds.
- A Flexible Batch Processing Scheduling (FBPS) strategy will be formulated to allow dynamic workload management through smart switching of batch and stream execution modes to enhance responsiveness, workload balancing, and resource utilization in varying cloud workloads.
- An Evolutionary Particle Genetic Optimizer (EPGO) is a system that is intended to optimize scheduling decisions by a multi-objective evolutionary optimization algorithm that seeks to minimize latency, maximize throughput, and balance the use of computational resources.
- A meta-orchestration system (MOS) is put in place to offer a secure governance layer, encrypted inter-module communication, real-time adaptive monitoring, and a secure runtime ledger (SRL) that are integrated to provide traceable, reliable, and autonomous orchestration of intelligent cloud pipelines.

This paper is further divided into the following sections. Part 2 presents both related works and the problem statement. The suggested method is implemented and illustrated in the part 3. The result and discussion are then presented in the part 4, followed by the conclusion in the part 5.

II. LITERATURE REVIEW

Wang *et al.*, [16] aimed to enhance personalized search by combining deep learning with cloud computing on AWS. With the support of a multi-layered adaptive model and three-tier monitoring, the presence of user preferences and context of user queries is obtained, with customized middleware for cloud configuration. Analysis of 100 million searches reveals that precision can be increased by 15%, cost-effectiveness improved by 12%, and low latency and scalable performance achieved. The results show enhanced accuracy, efficiency, and real-world deployment potential of AI-based search optimization.

Van *et al.*, [17] aimed to guide the technical implementation of AI pipelines for deep learning in imaging by addressing the design of standardized imaging biobanks and compliance with legal requirements. It underscores the importance of a hybrid strategy of using on-premise and cloud resources in order to address high data and computing demands. The methodology consists of mentioning the key ideas on the storage of data, integration of clouds, and architecture in an AI pipeline. These findings are supplemented with implementation messages that can be used in practice to optimize AI environments to train, validate, and deploy models.

Mohammed [18] examined the possibilities of optimizing cloud computing through AI, enhancing resource assignment, scalability, and facilitating the AI-as-a-Service (AIaaS). Due to mixed-method with industry perspectives, it concludes that AI implementation can decrease the cost of operations and downtime, and the value of the mid-sized enterprises through the AIaaS products. Nevertheless, issues like privacy of data, bias, and costs of integration are critical. The results indicate the possibilities of AI-cloud synergy and demand additional researches on these issues.

Kundavaram [19] optimized retail data pipelines to generative AI in the United States, focusing on such aspects as scalability, latency, and regulatory compliance. It relies on thematic analysis of secondary data to identify the best practices such as distributed computing, real-time monitoring, and secure data handling as a means to improve the reliability of pipelines. The results provide practical recommendations to enhance retailing processes by means of effective and legal generative AI processes.

Gudelli [20] attempted to optimize the performance of the AWS cloud systems by incorporating the AI-based assessment mechanisms. The research enhances the workload distribution, auto-scaling, and anomaly detection with the help of ML and predictive analytics to save money and increase productivity. Assessment of AI applications on Amazon computing services such as EC2, Lambda, and SageMaker demonstrates an impressive improvement in computation and cost efficiency. The results indicate the transformative effects of AI on the management of cloud resources as per the business needs.

Zheng *et al.*, [21] tried to make resource allocation in dynamic clouds as efficient as possible with the help of an AI-based hybrid predictive model of XGBoost and LSTM. It suggests a dynamic scaling algorithm based on the large-scale historical data, in which the prediction is taken into account together with the actual system states in the real time to distribute the available resources in a mature manner. The results of experiments show the use of more resources (65% to 83%), reduced proportion of the violation of the SLA in the process (2.5% to 0.8%), and efficiency in terms of energy. The model is superior to existing methods and is an all-inclusive solution to intelligent and autonomous cloud resources management.

Enemosah [22] discussed how AI-driven predictive models can be integrated into DevOps CI/CD pipelines to enhance efficiency, reliability, and scalability. The study predicts failures of weight structures, optimization of resource allocation, and deployment by employing the ML algorithms created and implemented through regression, clustering, neural networks, and reinforcement learning. Case studies of Jenkins and GitLab report the use of

drastic reduction in the build times and also the sprint of the success rate of deployments. The results highlight the contribution that AI can make to the priorization of the tests administered and their automatic monitoring, and in the process identify various obstacles, including the quality of the collected data and the organizational willingness to embrace AI.

A. RESEARCH GAP

The design and operation of intelligent data pipelines is a developing field of research, even though AI-powered cloud infrastructures have advanced rapidly. The previous research has been conducted on multiple aspects of AI-based cloud optimization. To illustrate, adaptive cloud schemes with deep learning models showed better performance with personalized services and efficiency of resources based on dynamic monitoring and scalable frameworks [16]. In a comparable vein, organized methods of AI pipeline design recognize the value of common infrastructure and hybrid cloud environments to embrace extensive computational courses [17]. Further studies have also identified the use of AI-based predictive models in cloud resource optimization, enhancing the scalability of the system and minimizing operational expenses in distributed systems [18–21]. Moreover, machine learning techniques used in DevOps pipelines have demonstrated the possibility of improving the reliability of deployments and automated system monitoring [22]. Though these contributions offer great information on AI-assisted cloud management, the majority of the current solutions cover the management of separate operational aspects of cloud management, such as scheduling, resource prediction, or monitoring. Therefore, the ability to implement a detailed architecture that will build adaptive pipelines, workload execution, and orchestration of secure orchestration is relatively under-researched in the present cloud computing studies. In contrast to the current AI-based cloud management tools, which mostly focus on standalone elements, like resource forecasting, workload orchestration, or workflow monitoring, the proposed framework presents a holistic architecture in which autonomous creation of cloud pipelines is produced through historical workflow analytics, and at the same time, adaptive scheduling, evolutionary optimization, and secure orchestration are introduced as part of a single intelligent system.

III. PROPOSED METHODOLOGY

To address the limitations of traditional cloud pipelines that lack adaptability and scalability, this research proposes an AI-driven approach for the automated generation of intelligent cloud architectures. The process starts with the gathering of historical workflow analytics, which are resource usage patterns, execution times, and system failure traces. These data are processed in advance to deal with the missing data and remove noise so as to have a clean reliable input to the process of pipeline generation. AMI engine then processes the refined data and determines behavioral trends and automatically creates optimized cloud structures. This allows the system to dynamically respond to the changing workload requirements as shown in Fig. 1.

Once the pipeline is generated, FBPS scheduling algorithm handles the workload scheduling, and the EPGO regularly adjusts the parameters of scheduling to achieve the best performance. The MOS is in charge of the overall execution, which is automation, consistency, and resilience. Finally, the assessment of the scalability, predictive precision, and efficiency of the system is carried out,

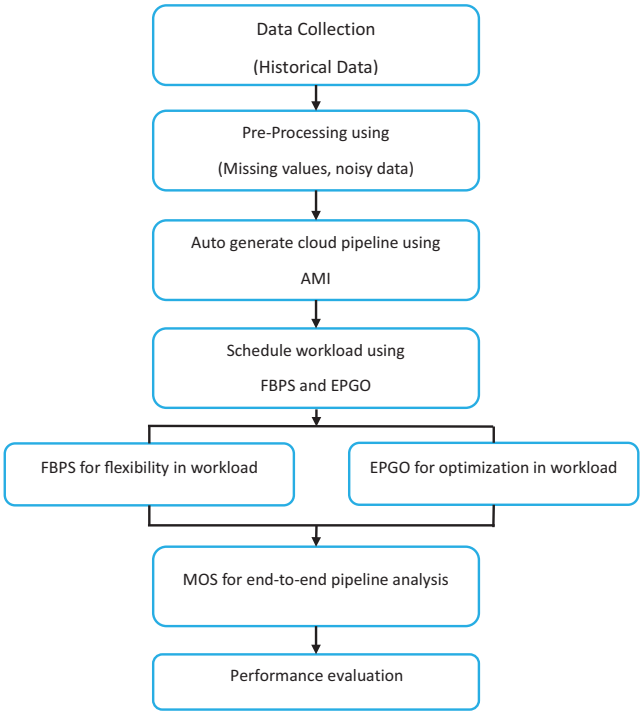

Fig. 1. Proposed AI-driven cloud pipeline architecture integrating AMI, FBPS, EPGO, and MOS.

which is consistent with the purpose of the study to develop AI-based cloud infrastructure.

A. DATA COLLECTION

In cloud computing, historical data contains information about the past performance of various activities. It outlines the resources in the system, the approach to carrying out tasks, and the challenges facing the system at different work times. It helps in building schedulers that are flexible and can evolve easily. Future developments and overlapping conflicts are noticed more easily by the intelligent systems, through the information they gather from the past. These systems are put together using the collected historical workflow data.

This research can access historical workflow data to analyze important metrics involved in managing the software pipeline. For this purpose, start time and end time are included in the history records to check how the job was processed in the past. The performance of using computer resources is judged by how much CPU and memory are being used. The efficiency of the scheduler can be determined by how quickly a task is completed and how long it takes to get attention. The system's performance is measured by examining how fast tasks are finished. Sharing the data sources and the number of steps in each task is required to confirm that all tasks are suitable for the situation. History gives an organization insight into systems, though it can often be disorganized, and more needs to be done with it. The data need to be structured and preprocessed before the analysis and cloud pipelines can take place.

B. PREPROCESSING

To ensure accurate measurement, completeness, and easy access for analysis, preprocessing needs to be made on the data before setting up smart systems. The cloud data is not always well ordered; thus, it

becomes difficult to tell what exactly the system is performing. Unless these problems are resolved, the development models, how much work needs to be done, and the choice of how to utilize resources will be more challenging. It runs certain rudimentary procedures initially to solve the problem in completing values and rumbling data.

1). MISSING VALUES. Missing entries in workflow logs can often be from network latency issues, sensor rigidity, collecting delays, or inconsistent time-stamping from distributed nodes. These deficiencies may frequently be observed in key parameters such as the time of task initiation, CPU usage, memory space allocation, and the duration of execution [23]. The lack of such information can cause the improper sequence of the pipeline phases and lead to the wrong interpretation of the system states.

One of the ways of fighting this challenge is to use a context-sensitive imputation. Interpolations of continuous variables are used to ensure that there are no sharp vibrations around the time series so as to keep the work behavior running naturally. With categorical attributes, for example, task status or type of data source, however, the value that is most frequently used in similar contexts is replaced to maintain the inferences of semantics that take place under the pattern of operation. In the event that the level of data absence is high and probability of imputing the missing values would increase the uncertainty in the concerned records, then those records are not incorporated into the follow-up analysis to ensure validity of results [24]. Such a selective imputation process is wasted to be sure that the dataset remains complete as well as representative to make possible reliable and precise pipeline analysis modules to be built.

2). NOISY DATA. Noise in the workflow logs frequently occurs as a result of sudden variations in workloads, invalid system log recordings, sensor errors, or as a passing anomaly in the recorded values. These irregularities increase the difficulty of interpreting execution behavior since the slight spikes or dips in measurements can confound patterns which actually exist. Therefore, it is harder to identify consistent trends of tasks and model pipelines and make them train and valid when noise is present.

In order to reduce these distortions, a two-step noise handling approach is employing. The former is the initial step of finding and eliminating abnormal values that are far beyond the anticipated values of operating ranges to ensure as minimal effect as possible of spurious CPU or memory activity, which could bias the trend analysis process [25, 26]. The second stage calls on the moving average smoothing to locate long-term tendency, as well as to fade the short-term changes. In such a manner, the workload dynamics underlying can be more effectively visualized and valid feature extraction toward following modeling is achievable.

Besides statistical filtering, semantic consistency laws are also applied in order to eliminate and rectify the logically irregular records, like tasks that are marked as 'finished' and with an execution time of 0. The errors are minimized step by step, and the outliers and atypical trends are normalized to provide the created data set to be with the control that is needed to automate the pipeline. The refined data not only bring an element of interpretability but also makes it possible to construct the cloud pipelines with above-average accuracy and robustness, and the pipelines to be constructed must be structured according to the working nature of each system.

C. AUTO-GENERATE CLOUD PIPELINES

The AMI algorithm, which takes on the challenge of building execution pipelines for microservices in the cloud, is designed to analyze historical workload patterns and system behaviors. Leveraging the structured product of past execution, AMI identifies the task interdependencies, resource utilization trends, and behavioral profiles to autonomously design and optimize the cloud architectures. The main goal is to optimize the pipeline management while providing the scalability, reproducibility, and adaptiveness to different heterogeneous workloads.

Pipeline generation starts with the retrieval of major parameters for each task in historical workflow data; parameters are the start and end time, time required to complete the task, CPU and memory use, the data source, as well as the task complexity. The current equation (1) is used to compute the execution time of a task T_i:

$$ExecutionTime_i = EndTime_i - StartTime_i \tag{1}$$

where $ExecutionTime_i$ denotes the execution time of the task i, $StartTime_i$ represents the start time of task i, and $EndTime_i$ represents the completion time of task i. To ensure timely scheduling and avoid execution bottlenecks, queuing time plays a key role in evaluating task urgency. As defined in equation (2), the queuing delay for a task T_i is computed as the difference between its actual start time and the time it was submitted to the system:

$$\mathrm{QueueTime_i} = \mathrm{StartTime_i} - \mathrm{SubmissionTime_i} \tag{2}$$

where $\mathrm{QueueTime_i}$ denotes the queuing delay of the task i, $\mathrm{StartTime_i}$ represents the execution start time, and $\mathrm{SubmissionTime_i}$ indicates the submission time of the task. Instead of depending on predefined structures, the AMI algorithm breaks down each task into gathering data, processing information, and saving it along the way. Using this information, AMI automatically creates the right pipeline modules. Because of this design, the algorithm can cope with a wide range of tasks in real-world situations and modify the pipeline to adjust to changes in the system.

In this phase, a DAG is built to demonstrate the order and connections between all the modules in the system. Each module in the DAG has a node, and the arrows between nodes represent which modules send information to and from each other. Using a graph makes it possible to see the order tasks are performed in, find areas where work is slowed down, and recognize elements that can be dealt with at the same time, all of which are beneficial for managing cloud resources effectively.

In the cloud pipeline architecture phase, resource allocation strategies and data routing policies are defined for each module. To guide intelligent decision-making, Using equation (3), AMI calculates the task utility score $U(T_i)$, which quantifies the computational significance of each task:

$$U(T_i) = \alpha \cdot \frac{CPU_i}{CPU} + \beta \cdot \frac{Mem_i}{Mem} + \gamma \cdot \frac{ExecTime_i}{ExecTime} \tag{3}$$

Here, α,β,γ are tuneable weighting coefficients reflecting system optimization priorities and $CPU_i, Mem_i, ExecTime_i$ represent the mean historical data of any task. The utility score is used to prioritize those tasks which are resource-intensive or performance-critical and to determine module isolation, resource quotas, and scheduling preferences.

As soon as the pipeline is fully configured, it is converted into a standard IaC format. Each module has a definition that describes its runtime, how it should be scheduled, and includes monitoring capabilities. Having an encoded template with a Helm chart in place allows Kubernetes or Terraform to deploy the pipeline smoothly in any environment.

Algorithm 1. AMI

Input:

D = Historical Workflow Dataset, where each task t_i has: $\{Start\_Time, End\_Time, Submission\_Time, CPU\_Usage, Memory, Task\_Dependencies\}$

Output:

G = Optimized Cloud Execution Pipeline (DAG + IaC Configuration)

Initialize Graph $G = \emptyset$
For each task $t_i \in D$ do
    Compute $Execution\_Time\ E_i = End\_Time - Start\_Time$
    Compute $Queue\_Delay\ Q_i = Start\_Time - Submission\_Time$
    Compute $Task\_Utility\_Score\ U_i$ using Equation (1)
End For
Cluster tasks into behaviour profiles based on $\{E_i, CPU\_Usage, Memory\}$
For each cluster C_j do
    If C_j performs ingestion:
        Create Module $M_i = Data\_Ingestion\_Module$
    Else if C_j performs computation:
        Create Module $M_i = Processing\_Module$
    Else if C_j requires persistent state:
        Create Module $M_i = State\_Storage\_Module$
    Add M_i to G
End For
For each dependency $(t_i = t_j) \in D$ do
    Add $Directed\_Edge(M_i = M_j)$ to G
End For
For each Module $M_i \in G$ do
    Assign $Resource\_Quota = \{CPU, Memory\}$ based on U_i
    Define $Data\_Routing\_Policy\ and\ Scheduling\_Trigger$
    Generate $IaC\_Config\_Block(M_i)$ in Helm and YAML format
End For
During Runtime:
    Monitor Metrics = {Latency, Throughput, Utilization}
    If deviation from historical baseline detected:
        Reconfigure Module Parameters or Reschedule
    End If
Return Final Encoded Pipeline G

It further uses a feedback integration process to help the system adjust during runtime. During the execution of a pipeline, real-time metrics on latency, number of products made, and system demand are compared to past systems' average performance. If any challenges are uncovered, the configuration rules for the pipeline are altered. Thanks to the feedback loop, the system keeps adjusting to new and changing demands in the workload. As time goes on, AMI can review its actions and adjust its process for generating new opportunities.

In this strategy, the AMI algorithm is able to automatically produce intelligent cloud pipelines. By being both modular, fast, and reliable, it supports the large-scale and smart handling of workloads in today's cloud computing market.

The AMI algorithm automates the generation of cloud execution pipelines by learning from historical task behaviors and resource usage. It eliminates the need for manual configuration through modular abstraction and behavior-driven task clustering. This makes it possible to have scalable, reusable, and adaptive pipelines, in response to changing workload patterns.

D. WORKLOAD SCHEDULING

Efficient workload scheduling plays a pivotal role in ensuring the timely, balanced, and resource-aware execution of tasks within automated cloud pipelines. Due to the heterogeneity of the cloud environment and the dynamically varying rates of arrival of tasks and variability of the computational demands, the static or monolithic strategy of scheduling is not enough. We need a more dynamic and smart scheduling system instead, one that can respond dynamically to changes in workload, priority of tasks, and system load changes. This paper presents a two-layer, modular workload scheduling system, in which the former layer is focused on the flexibility of the system, whereas the latter aims at optimization.

1). FBPS. The FBPS scheduling algorithm introduces real-time adaptability to the cloud task scheduling frameworks. The primary role is that it dynamically switches between batch-mode and stream-mode execution as the workload intensity and task size change and system utilization. This flexibility is essential in heterogeneous environments where workloads change not only

in quantity but also in frequency and priority, and such fixed policies dealing with scheduling often may not be effective in providing efficiency or equity.

Each incoming task in the system is analyzed once it arrives, and various features are extracted for classification, such as the workload size, priority, and arrival time. If a task is delay-tolerant or performs a labeling exceeding a defined pool threshold of workload, it is attributed to batch type, and the remainder fall as stream type. The workload scheduling then reacts accordingly: while there is moderate system load, the batch-type tasks are grouped together and sent to the compute cores with the least load so that maximum resource utilization is achieved. In contrast, under high system stress or erratic arrival patterns, tasks are assigned in stream mode, that is, to the first available core, so that it can also guarantee responsiveness to bursty and pathogenic workloads.

A module for dynamic reassessment is included in FBBS so that it can replan assignments for each core every fixed period. The module checks the system from time to time to see which compute resources are busiest or least used and moves assigned yet uncompleted tasks between them accordingly. To support this decision-making process, FBPS computes a Task-Centric Load Index (TLI) for each task core pair:

$$TLI(T_i,C_j) = \lambda_1 \cdot \frac{Size_i}{Capacity_j} + \lambda_2 \cdot Priority_i + \lambda_3 \cdot \frac{WaitTime_i}{AvgWait_j} \quad (4)$$

In this equation (4), T_i refers to the i^{th} task and C_j to the j^{th} compute core. The variables $Size_i$, $Priority_i$, and $WaitTime_i$ represent the task's workload size, its normalized priority, and time spent in the queue, respectively. $Capacity_j$ denotes the computational capability of the target core, and $AvgWait_j$. The average queueing delay associated with that core. The coefficients are $\lambda_1,\lambda_2,\lambda_3$. These are tunable weights that balance throughput, priority sensitivity, and fairness.

FBPS provides responsive management of workloads by using adaptive scheduling with real-time system information in the cloud. Because of this, pipelines perform well in many situations and become an essential part of the overall orchestration system. With the help of AIM, the cloud infrastructure keeps many different tasks running smoothly and fast, ensuring resources are allocated fairly, which is important for the intelligent and automatic execution of the pipeline.

FBPS introduces real-time flexibility in scheduling by dynamically switching between batch-mode and stream-mode execution. It accounts for task priority, workload size, and core availability to optimize throughput and reduce queuing delays. This enhances the responsiveness and fairness of cloud pipeline execution.

2). OPTIMIZATION ALGORITHM EPGO. To achieve long-term scheduling efficiency and adaptability of the cloud pipeline, in the given study a lightweight evolutionary optimizer, abbreviated as EPGO, is used. This optimizer keeps the parameters that control the scheduling of workloads (optimal batch sizes, resources-allocation thresholds and priority weightings) in tune. Such dynamic changes enable the scheduling layer to react to both instantaneous system conditions and dynamic changes in workload behavior.

The EPGO has a set of candidates scheduling settings on which it works. There is a set of not repeated parameters of scheduling that are determined by each candidate. The following configurations are optimized by a multi-objective fitness function in the course of every iteration: The performance measures

Algorithm 2. FBPS

Input:
Q = Incoming Task Queue
C = Set of Compute Cores $\{c1,c2,\ldots,c_m\}$
Output:
S = Optimized Scheduling Map of Tasks = Cores
For each task $t_i \in Q$ do
  Extract Features = $\{Worklo\_Size\ w_i, Priority\ p_i, Arrival\_Time\ a_i\}$
  Classify t_i as:
    Batch_Type if $w_i \geq$ Threshold or has delay tolerance
    Stream_Type otherwise
End For
If *Current_System_Load* $\leq$ *Load_Threshold* then
  Aggregate *Batch_Type* Tasks into Buffer B
  If $|B| \geq Max\_Batch\_Size$:
    Assign B to least-loaded cores in C
  End If
Else
  For each *Stream_Type task* t_j:
    Assign t_j to earliest available core in C
  End For
End If
Every ΔT time units do
  For each core $c_k \in C$ do
    Compute Task-Centric Load Index (TLI_{ik}) using Equation (2)
    If Load Imbalance Detected:
      Migrate queued tasks to underloaded cores
    End If
  End For
End Every
Return Scheduling Map S

included in the definition are task throughput, latency, and resource load balancing:

$$F(P_i) = \omega_1 \cdot \frac{Throughput}{MaxThroughput} - \omega_2 \cdot \frac{Latency}{MaxLatency} - \omega_3 \cdot \frac{Imbalance}{MaxImbalance} \quad (5)$$

In equation (5), where $F(P_i)$ denotes the fitness value of configuration P_i, *Throughput* represents the number of tasks processed, *Latency* denotes the average task delay, and *Imbalance* indicates workload imbalance across computing resources. *MaxThroughput*, *MaxLatency*, and *MaxImbalance* are normalization constants. The parameters ω_1, ω_2, and ω_3 are weighting coefficients controlling the importance of throughput, latency, and load balance.

Each time the algorithm optimizes, it examines all the configurations and selects the best ones based on their scores. It brings each candidate up to date using its best local information and the best solution found around the world. Small changes and variations are introduced to help explore other possible settings. It combines parts of identical-to-best designs with unique features. Should these

Algorithm 3. EPGO

Input:

P = Initial Population of Scheduling Configurations $\{\theta_1, \theta_2, \ldots, \theta_n\}$

$F(\theta)$ = Fitness Function combining: Throughput (T), Latency (L), Resource Balance (R)

Output:

θ^* = Globally Optimal Scheduling Configuration

For each candidate $\theta_i \in P$ do
    Initialize $personal\,Best_i = \theta_i$
End For
$globalBest = argmax\_\theta_i \in P[F(\theta_i)]$
For iter = 1 to $Max\_Iterations$ do
    For each candidate $\theta_i \in P$ do
        Evaluate Fitness: $F(\theta_i) = \alpha_1 \cdot T - \alpha_2 \cdot L + \alpha_3 \cdot R$
        If $F(\theta_i) > F(personlBst_i)$:
            $personalBest_i = \theta_i$
        End If
        If $F(\theta_i) > F(globalBest)$:
            $globalBest = \theta_i$
        End If
    End For
    For each $\theta_i \in P$ do
        Update $\theta_i = \theta_i + \beta_1(personalBest_i - \theta_i) + \beta_2(globalBest - \theta_i)$
        Apply Mutation and Crossover to promote diversity
    End For
    Select two candidates θ_a, randomly
    Generate Offspring $\theta_{new} = Blend(\theta_a, \theta_b) + \varepsilon(noise)$
    If $F(\theta_{new}) > F(worst\ in\ P)$:
        Replace worst with θ_{new}
    End If
End For
Return Optimal Configuration $\theta^* = globalBest$

new solutions perform better than the old ones, they will be adopted by the population, and the system will improve its scheduling.

As the optimization continues, it often gives the FBPS scheduling algorithm updated parameters from its optimal solutions. The way the system works allows it to change by itself, responding to current needs and also to previous shifts in workload to promote efficiency. When FBPS and EPGO are used, the cloud pipeline scheduling framework can adjust to changing environments in the cloud and keep improving its performance.

To optimize scheduling parameters, EPGO constantly adjusts its parameters through an evolutionary optimization process controlled by a multi-objective fitness measure. It allows the work balance, latency, and resource usage to be controlled adaptively. It is a continuous improvement in FBPS settings by refining it until performance is continually enhanced in dynamic settings.

E. END-TO-END PIPELINE ANALYSIS

In contemporary AI-enabled cloud pipelines, dynamic workload fluctuation, scattering planning, and multi-module control necessitate a strong and smart orchestration structure. In response to this requirement, we have envisioned the MOS as a new, security-aware, and self-adaptive orchestrator to connect the intelligent scheduling through FBPS with optimization using EPGO so that the end-to-end governance of cloud pipeline execution is achieved.

1). MOS UNIQUELY INTEGRATES. MOS coordinates the three main capabilities in an orchestration layer

(i) **Secure Module Communication:** It is also guaranteed that there is no communication among the pipeline modules without transiting over an encrypted, authenticated channel that maintains the confidentiality and integrity of the critical parameters, such as the optimized decision-making forward schedule and status of the operations that are to be performed.

(ii) **Real-Time Adaptive Monitoring:** MOS constantly monitors important performance numbers through execution time of tasks (wait time: W_t), percentage of tasks (throughput: T_p), and percentage of resources utilized (resource usage: R_u). Provided that deviations or anomalies are identified, MOS initiates the corrective actions automatically–either by updating the parameters to FBPS or making re-optimization requests to EPGO.

(iii) **Auditable Runtime Ledger:** A SRL captures every action, configuration, and every response of the system. This allows auditing within a version-controlled replay, and audits after execution, as well as forensic debugging capabilities not normally provided in standard orchestration platforms.

This automation, protection, and traceability triple combine make MOS superior to traditional orchestrators, but more of an intelligent layer of governance that can self-optimize continuously and enforce security and policy compliance in multi-tenant environments.

2). SECURE AND ADAPTIVE FUNCTIONALITY. The transmission of data between modules located on the pipeline is also encapsulated under a secure data transmission model in algorithm 4 which is provided as:

$$SecureComm(M_i \leftrightarrow M_j) = Enc_k(Paload_{i \to j}) \tag{6}$$

In equation (6), where $SecureComm(M_i \leftrightarrow M_j)$ denotes the secure communication established between pipeline modules M_i and M_j, $Enc_k(\cdot)$ represents the encryption operation performed using the symmetric encryption key k, and $Payload_{i \to j}$ denotes the data or message transmitted from the module M_i to module M_j. This mechanism guarantees that scheduling configurations, optimization feedback, and telemetry data remain confidential and tamper-proof.

MOS evaluates the pipeline based on metrics during its run time. When a performance deviation like a high wait task has been detected, a response action takes place in equation (7):

$$If\ W_t > \tau\ then\ Trigger\ the\ Reoptimize\ using\ EPGO \tag{7}$$

where W_t denotes the observed task waiting time at time t, and τ represents the predefined threshold that defines the acceptable waiting-time limit for tasks in the scheduling queue. This enables online modulation of scheduling or optimization parameters, thus ensuring the system operates dynamically with respect to changing loads.

Each orchestration cycle is logged in the SRL using equation (8):

$$SRL_t = \{Task_i, Config_t, Decision_t, Metrics_t\} \tag{8}$$

where SRL_t represents the runtime ledger entry recorded at time t, $Task_i$ denotes the task identifier corresponding to the pipeline operation being executed, $Config_t$ indicates the configuration

Algorithm 4. MOS

Input:

$M = \{\text{AMI, FBPS, EPGO}\}$

Metrics = $\{W_t, T_p, R_u\}$

SRL = Secure Runtime Ledger (empty)

Output:

Orchestrated, secure, and auditable pipeline execution

Initialize secure communication channels for all $M_i \in M$

For each module pair $(M_i \leftrightarrow M_j)$:

    Transmit $Payload\{i \to j\} = Enc_k(Payload_{i \to j})$

End For

While pipeline execution is active:

    Monitor metrics: $\{W_t, T_p, R_u\}$

    Log entry to $SRL_t = \{Task_i, Config_t, Decision_t, Metrics_t\}$

    If $W_t > \tau$ or performance drift detected:

        If scheduling delay → update FBPS parameters

        If resource imbalance → invoke EPGO re-optimization

    End If

End While

Persist SRL and final state for audit/replay

Return status: Secure, adaptive, traceable execution complete

parameters applied to the pipeline at the time t, $Decision_t$ represents the orchestration or scheduling decisions generated by the MOS at that time, and $Metrics_t$ represents the observed system performance metrics including the execution latency, throughput, and resource consumption. This hierarchical logging can be used to monitor transparently, execute with traceability, and audit reproducibly in the pipeline cloud operations in a distributed environment.

The MOS offers a novel, unified framework that ensures secure, adaptive, and auditable execution of intelligent cloud pipelines. It is the only system to combine encrypted communication, real-time performance tracking and traceable runtime ledger. This allows independent coordination between the modules of scheduling and optimization without the involvement of manpower. Consequently, MOS introduces a new benchmark of secure, autonomous orchestration of dynamic, multi-tenant cloud environments.

3). SECURITY ANALYSIS OF MOS. The proposed MOS is security-conscious at the orchestration layer of the cloud pipeline architecture, as opposed to traditional orchestration systems where security controls and offerings are mostly at the infrastructure layer. The traditional orchestration models are mostly focused on container deployment, orchestrating workflows, and managing resources, and security protection remains the responsibility of additional infrastructure components, such as firewalls, access control, or encryption at the network level. However, unlike this, MOS also uses encrypted inter-module communication, adaptive runtime monitoring, and SRL as part and parcel of the process of orchestration itself. This design gives safety to sensitive parameters of the orchestration, scheduling choices, and optimization feedback during execution and still gives long-term visibility of system behavior. In addition, a ledger-based logging system allows the transparent auditing, traceability, and reproducibility of pipeline activities in a distributed environment. MOS offers stronger protection, monitoring, and accountability by integrating security, monitoring, and accountability into the orchestration workflow, which ensures greater protection, greater knowledge of operations, and successful coordination of AI-based cloud pipeline management systems.

F. COMPUTATIONAL COMPLEXITY ANALYSIS

The complexity of the proposed framework computation can be determined with respect to the combination of the AMI, FBPS, EPGO, and MOS modules. The AMI module builds the pipeline structure using historical workflow tasks analysis and produces a DAG, whose time complexity is O(N + E), where N is the number of tasks, and E is the tasks' dependency. The FBPS scheduling algorithm allocates tasks in dynamically computed resources of complexity of O(N). The EPGO optimizer can be executed over a population of candidate solutions whose complexity is O(P × I), where P and I are the population size and optimization iterations, respectively. Lastly, the MOS orchestration layer conducts monitoring and logging with complexities of O(M), and M is the number of pipeline modules, hence scalable performance with large cloud workloads.

AMI pipeline generation from historical workflows is automated, FBPS real-time adaptability is dynamic scheduling, and EPGO task allocation optimization is evolutionary refining. Collectively, the objectivity of these techniques creates a secure, scalable, and adaptive structure to ensure a highly effective and efficient cloud pipeline execution and ensures that system complexity is in place to meet changing and diverse computational needs.

IV. RESULT AND DISCUSSION

A. DATASET DESCRIPTION

The data utilized in this work are based on the NSL-KDD publicly available network intrusion detection dataset, which was obtained from Kaggle [27]. The NSL-KDD dataset has been derived from the popular KDD Cup 1999 dataset and is popularly used to test intelligent security and workload analysis systems. It records organized documentation of network traffic behavior, such as normal activities and other forms of abnormal occurrences. The data is composed of some 125,973 training and 22,544 testing records, each instance of which has 41 feature attributes and a target label. These properties consist of traffic statistics, which include the type of protocol, the type of service, the duration of connection, the number of packets, and error rates, which are all products of system behavior under varying conditions of operation. Since the dataset is reflective of real-world network workload profiles, it offers a valid reference point in the assessment of intelligent scheduling and monitoring systems in distributed contexts.

Within the framework of the present research, the NSL-KDD dataset is used to emulate the historical workflow analytics to execute the cloud pipeline. Instances of every dataset are handled as distinct workflow activities, whereas features are indicators of system performance like workload intensity, resource demand, and operational behavior. By considering these parameters, the proposed framework could help analyze the nature of tasks, project behavioral patterns, and create the most optimized cloud pipeline architecture using the AMI module. This data format, hence, facilitates the tests of adaptive scheduling using FBPS and optimization utilizing EPGO while enabling MOS to assess conditions of execution and identify abnormalities. It is also due to the use of a publicly available dataset that the proposed framework can be

reproducible in an experiment, with other researchers being able to validate it and compare its performance to the existing cloud scheduling and optimization strategies.

B. COMPARISON OF THE PROPOSED AI-DRIVEN CLOUD PIPELINE ARCHITECTURE WITH THE EXISTING MODELS

In this comparison, the proposed AI-driven cloud pipeline architecture is compared with the existing models such as Hybrid Parallel Enhanced Genetic Algorithm (HPEGA), Particle Swarm Optimization (PSO), Hybrid Genetic Algorithm (HGA), and Genetic Algorithm (GA) [28] based on Performance matrix, convergence analysis, model accurate prediction matrices, scalability and throughput, execution efficiency, and CPU and memory utilization.

1). PERFORMANCE MATRIX. A comparison of accuracy, precision, recall, and F1-score for the local maximum is shown in Fig. 2, along with the scores from the four baseline optimization techniques: HPEGA, PSO, HGA, and GA. All the evaluated aspects show that the proposed AI-driven cloud pipeline architecture outperformed, providing an accuracy of 0.968, a precision of 0.962, a recall of 0.959, and an F1-score of 0.960, demonstrating that it is accurate and consistent in handling any situation. Instead, the HPEGA approach, which comes in second place, results in relatively poorer readings (accuracy: 0.935, F1-score: 0.923). PSO performs relatively well, followed by HGA and GA, with the worst results (0.876 accuracy, 0.862 F1-score) due to its struggle to adapt to the project's various demands. The findings prove that the suggested AMI-FBPS-EPGO mechanism excels in carrying out intelligent cloud pipelines in a reliable, timely, and performance-driven way.

2). CONVERGENCE ANALYSIS. Figure 3 shows the proposed AI-driven cloud pipeline architecture versus HPEGA, PSO, HGA, and GA after 50 iterations. The value for fitness is seen on the y-axis, and the number of iterations is on the x-axis. The proposed method was able to get very close to the optimal solution much faster than the other algorithms. After about 20 generations, the approach reaches a relatively low fitness, which demonstrates that it converges fast and is capable of effective optimization. Unlike the other methods, the GA and HGA converge slowly and leave higher residual fitness values, which may mean that global search is weaker and takes longer to complete. Because the proposed AI-driven cloud pipeline architecture steadily improves and then stabilizes quickly, it proves itself effective at finding optimal schedules without taking too long to compute.

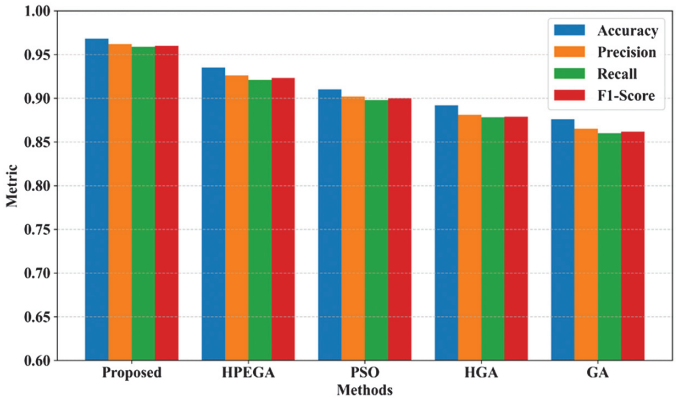

Fig. 2. Performance comparison of the proposed AI-driven cloud pipeline architecture with HPEGA, PSO, HGA, and GA based on accuracy, precision, recall, and F1-score.

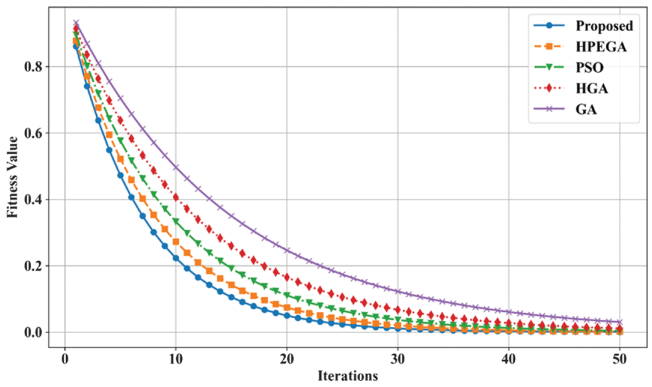

Fig. 3. Convergence analysis of the proposed AI-driven cloud pipeline architecture compared with HPEGA, PSO, HGA, and GA across iterations.

3). MODEL-ACCURATE PREDICTION MATRICES. Figure 4 shows the MAPE for each of the prediction models compared. The model has the best results, as it achieves the least MAPE (4.1%). In a contrasting way, models HPEGA, PSO method, HGA, and GA produce errors more frequently. This suggests that the suggested modeling technique is more accurate than the previous techniques for the same task.

4). SCALABILITY AND THROUGHPUT. Figure 5 compares the scalability and throughput of five algorithms: Proposed, HPEGA, PSO, HGA, and GA, based on accuracy (%) in terms of growing sizes of the dataset (30% to 100 %). The proposed AI-driven cloud pipeline architecture consistently performs better than all others and reaches 92.5% when the entire dataset is used (HPEGA obtains 88%, PSO 85%, HGA 83%, and GA 82%). This is the indication of very high scalability and throughput of the proposed AI-driven cloud pipeline architecture, where even in the increasing data volume, the performance remains high. It demonstrates an average of 510% improvement in accuracy when compared to other models in all the dataset sizes, which confirms its strength and performance when in a large-scale cloud-based AI pipeline setting.

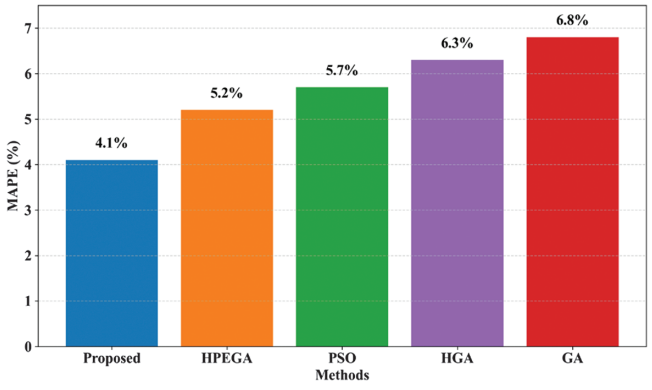

Fig. 4. MAPE comparison between the proposed AI-driven cloud pipeline architecture and existing optimization algorithms.

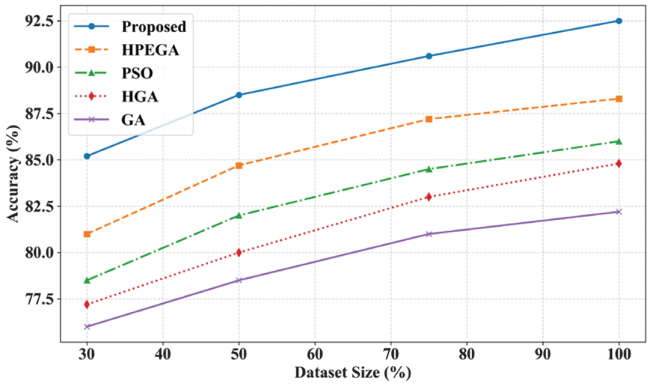

Fig. 5. Scalability and throughput comparison of the proposed framework with existing algorithms under different dataset sizes.

5). EXECUTION EFFICIENCY. Figure 6 compares the timings necessary to execute a task (ms) and generate the pipeline (sec) of five algorithms. The proposed AI-driven cloud pipeline architecture exhibits the best level of efficiency in execution, the least value of task execution time (∼150 ms), and pipeline generation time (∼10 sec). Compared to it, GA has the worst performance with a running time of ∼220 ms and ∼18 sec to generate the pipeline time. It means that the proposed system requires 32% less time to execute the tasks and 44% less time to generate the pipeline as compared to GA. On the whole, the proposed AI-driven cloud pipeline architecture is capable of bringing a 25–35% increase in efficiency of execution compared to current approaches, which means quicker and more responsive working of the pipeline.

6). CPU AND MEMORY UTILIZATION. Figure 7 compares the CPU utilization (%) of five algorithms, which were Proposed, HPEGA, PSO, HGA, and GA. The proposed AI-driven cloud pipeline architecture has the least median CPU usage (∼48%) with a small difference, pointing to optimal processing and a low number of computational overheads. On the contrary, the median utilization values of other algorithms, such as HPEGA, PSO, and GA, are increased between 57% and 62%, meaning that interquartile ranges are also expanded, and outliers exceeding 70% utilization values appear. In comparison to the algorithm which consumed, the most the CPU (GA at ∼69%), proposed AI-driven cloud pipeline architecture has a 30% lower CPU consumption than the one mentioned, and compared to its counterparts, it consumes,

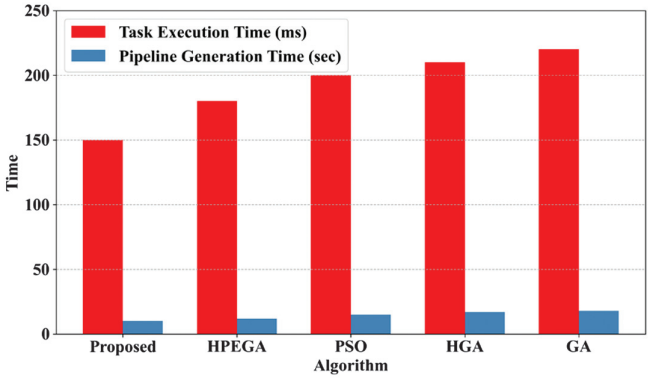

Fig. 6. Execution efficiency comparison showing task execution time and pipeline generation time.

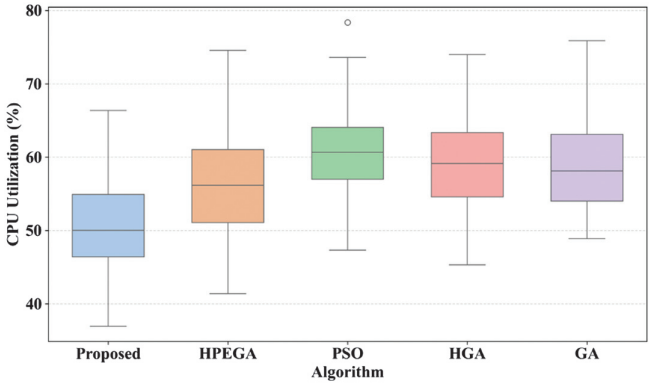

Fig. 7. CPU utilization comparison of the proposed AI-driven cloud pipeline architecture and existing algorithms.

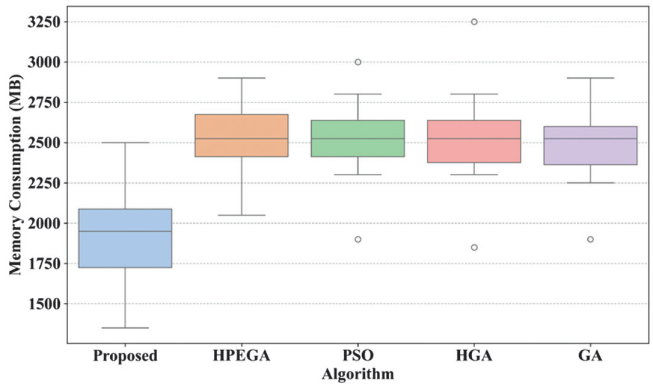

Fig. 8. Memory utilization comparison between the proposed AI-driven cloud pipeline architecture and baseline algorithms.

on average, 18–25% less CPU than those, which makes it a much more efficient and scalable algorithm in a computation-sensitive setting.

Figure 8 shows a comparison of the memory usage (in MB) of five algorithms: Proposed, HPEGA, PSO, HGA, and GA. The proposed AI-driven cloud pipeline architecture has the minimum median memory allocation (∼2000 MB) and interquartile range, which are signs of efficiency and reliability in its performance. In comparison, memory usage is greater and more spread out with HGA and PSO, with spikes at around 3250 MB. The proposed AI-driven cloud pipeline architecture decreases the amount of memory used by about 38.5% compared to that of the highest-consuming algorithm (HGA). The proposed system is also less demanding on memory with an average of 22–30% compared to its counterparts, indicating that the system will be adaptable to resource-limited setting that keep a high demand on memory.

C. COMPUTATIONAL COMPLEXITY ANALYSIS

Figure 9 shows the behavior of the proposed computational complexity of the AMI–FBPS–EPGO–MOS framework by examining the execution time with increasing numbers of workflow tasks. As the size of workload doubles (100–1000 tasks), the time spent performing the task increases at a slower pace, attaining around 2.3 seconds up to 11.8 seconds, and exhibits a controlled computational growth. The framework captures the execution time

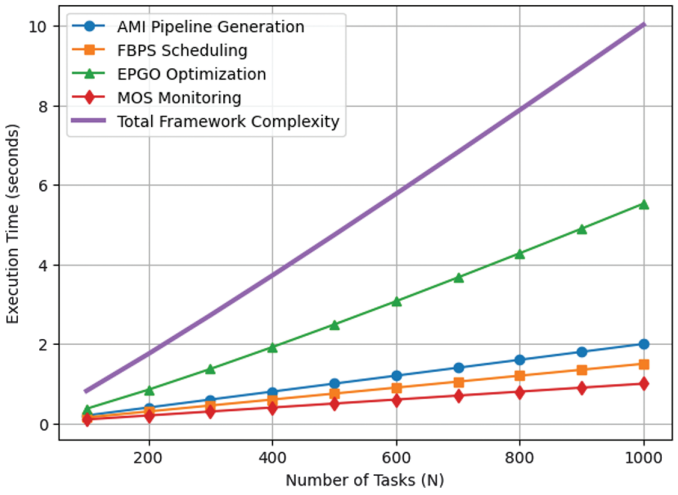

Fig. 9. Computational complexity analysis of the proposed AMI–FBPS–EPGO–MOS framework with increasing number of tasks.

of approximately 5.6 seconds and 8.9 seconds at medium workload levels of 400 and 700 tasks, respectively, which means that it is stable in terms of scalability. Such a slow growth is an indication that the combined scheduling, optimization, and orchestration systems can work effectively with little to no computational cost involved. The findings indicate that the given framework ensures a high level of performance and scalability of large-scale AI-based cloud pipeline architecture.

D. DISCUSSION

The presented AI-based framework has incredible performance characterized by a row of confirmed experimental results. It performs the best with the highest accuracy of 92.5% and 5–10% improvements over conventional models and even the lowest MAPE of 4.1%, which proves the reliability in predicting performance. Within 20 iterations, the system converges quickly and speeds up execution by 32% of the task completions and 44% of the pipeline generation. It remains consistent in its accuracy as the dataset size grows; hence, its accuracy is scaled and throughput. Also, it is 18–30% more CPU-specific and up to 38.5% memory-efficient as compared with competing approaches, thereby being highly applicable in computation-sensitive and resource-tightened surroundings. These extensive findings support the validity and flexibility of the model with regard to its practicality in the real-world intelligent, scalable cloud pipeline management.

V. CONCLUSION

The AI-driven cloud pipeline architecture, which incorporates AMI, FBPS, EPGO, and MOS, demonstrates unprecedented performance across all evaluation attributes. The framework efficiently solves the resource utilization and the absence of adaptive scheduling and secure orchestration mechanisms, by enabling intelligent pipeline generation, dynamic workload management, and coordinated system governance within a unified framework. Experimental results further validate the effectiveness of the proposed framework, achieving a low MAPE of 4.1%, a maximum accuracy of 92.5%, and rapid convergence within only 20 iterations. It also overcomes the baseline models in the reduction of task execution time by 32% and pipeline generation time by 44%, making it have an efficiency boost of up to 25–35%. Also, the proposed framework demonstrated improved scalability, preserving high levels of accuracy as the size of the dataset increased, and demonstrated resource efficiency used 18–30% less of the CPU and 22–38.5 % less of the memory than other solutions. Such results confirm its real-life application in time-sensitive, massive data AI implementations in the fields of healthcare analytics, financial projections, and automation. The future work will generalize the system to hybrid and multi-cloud implementation, combine real-time failure prediction, and demonstrate its resilience against a variety of real-world production data.

FUNDING

On behalf of all authors, the corresponding author states that they did not receive any funds for this project.

CONFLICTS OF INTEREST STATEMENT

The author(s) declare that they have no conflicts of interest to report regarding the present study.

AUTHOR CONTRIBUTIONS

All authors significantly contributed to the development of the described tool and are currently actively involved in it. The first draft of the manuscript was written by corresponding author, and all authors improved on previous versions of the manuscript. All authors read and approved the final manuscript.

REFERENCES

[1] S. Priyadarshini *et al.*, "Enhancing security and scalability by AI/ML workload optimization in the cloud," *Cluster Comput.*, vol. 27, no. 10. pp. 13455–13469, 2024.

[2] P. R. D. Achanta, "The Role of Artificial Intelligence in Shaping the Future of Cloud Technologies Benefits and Barriers," In 2025 Global Conference in Emerging Technology (GINOTECH) (pp. 1–8), IEEE, 2025.

[3] A. Shahidinejad, M. Ghobaei-Arani, and M. Masdari, "Resource provisioning using workload clustering in cloud computing environment: A hybrid approach," *Cluster Comput.*, vol. 24, no. 1, pp. 319–342, 2021.

[4] A. Hussain, "AI-Enhanced Cybersecurity: Streamlining Data Pipelines and Fortifying Cloud Infrastructure in a Digital-First Era," 2024.

[5] X. Wang, "Dynamic Scheduling Strategies for Resource Optimization in Computing Environments," 2024. arXiv preprint arXiv:2412.17301.

[6] P. B. Desai and O. Goel, "Scalable data pipelines for enterprise data analytics," *Int. J. Res. All Subj. Multi Lang.*, vol. 13, no. 1, p. 174, 2025.

[7] A. Sadykova, "Artificial intelligence for automated data workflow optimization in cloud-based big data systems," *Transdiscipl. Adv. Soc. Comput. Complex Dyn., Comput. Creat.*, vol. 14, no. 10, pp. 12–22, 2024.

[8] M. Trigka and E. Dritsas, "Edge and cloud computing in smart cities," *Future Internet*, vol. 17, no. 3, p. 118, 2025.

[9] S. Gulati, A. Tyagi, and P. K. Goel, "Security automation and orchestration in the cloud," *Analyzing and Mitigating Security Risks*

in Cloud Computing, P. K. Goel, H. M. Pandey, A. Singhal and S. Agarwal (Eds.), Hershey, Pennsylvania, USA: IGI Global Scientific Publishing, 2024, pp. 24–47.

[10] A. Quemy, "Two-stage optimization for machine learning workflow," *Inf. Syst.*, vol. 92, p. 101483, 2020.

[11] P. Nawrocki, M. Grzywacz, and B. Sniezynski, "Adaptive resource planning for cloud-based services using machine learning," *J. Parallel Distrib. Comput.*, vol. 152, pp. 88–97, 2021.

[12] N. Rana *et al.*, "A hybrid whale optimization algorithm with differential evolution optimization for multi-objective virtual machine scheduling in cloud computing," *Eng. Optim.*, vol. 54, no. 12, pp. 1999–2016, 2022.

[13] S. Nabi, M. Ibrahim, and J. M. Jimenez, "DRALBA: Dynamic and resource aware load balanced scheduling approach for cloud computing," *IEEE Access*, vol. 9, pp. 61283–61297, 2021.

[14] L. Wratten, A. Wilm, and J. Göke, "Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers," *Nat. Methods*, vol. 18, no. 10, pp. 1161–1168, 2021.

[15] J. Hviid *et al.*, "AI Pipelines: A Scalable Architecture for Dynamic Data Processing," In 2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C) (pp. 85–93). IEEE. 2025.

[16] J. Wang *et al.*, "Enhancing personalized search with AI: A hybrid approach integrating deep learning and cloud computing. *J. Adv. Comput. Syst*., vol. 4, no. 10, pp. 1–13, 2024.

[17] P. M. Ooijenvan, E. Darzidehkalani, and A. Dekker, AI technical considerations: Data storage, cloud usage and AI pipeline. arXiv preprint arXiv:2201.08356, 2022.

[18] K. Mohammed, "AI in cloud computing: Exploring how cloud providers can leverage AI to optimize resource allocation, improve scalability, and offer AI-as-a-service solutions," *Adv. Eng. Innov.*, vol. 3, pp. 22–26, 2023.

[19] V. N. K. Kundavaram, "Optimizing data pipelines for generative AI workflows: Challenges and best practices," *IJSAT-Int. J. Sci. Technol.*, vol. 16, no. 1, 2024.

[20] V. R. Gudelli, "AI-powered insights for performance optimization in AWS cloud environments," *Int. J. Sci. Res Appl.*, vol. 10, no. 2, pp. 1267–1276, 2023.

[21] H. Zheng *et al.*, "Efficient resource allocation in cloud computing environments using AI-driven predictive analytics," *Appl. Comput. Eng.*, vol. 82, pp. 17–23, 2024.

[22] A. Enemosah, "Enhancing DevOps efficiency through AI-driven predictive models for continuous integration and deployment pipelines," *Int. J. Res Publ. Rev.*, vol. 6, no. 1, pp. 871–887, 2025.

[23] A. E. Karrar, "The effect of using data pre-processing by imputations in handling missing values," *Indones. J. Electr. Eng. Inf. (IJEEI)*, vol. 10, no. 2, pp. 375–384, 2022.

[24] L. Yu *et al.*, "Missing data preprocessing in credit classification: One-hot encoding or imputation?" *Emerg. Mark. Fin. Trade*, vol. 58, no. 2, pp. 472–482, 2022.

[25] E. Cho, T. W. Chang, and G. Hwang, "Data preprocessing combination to improve the performance of quality classification in the manufacturing process," *Electronics*, vol. 11, no. 3, p. 477, 2022.

[26] T. Emmanuel *et al.*, "A survey on missing data in machine learning," *J. Big Data*, vol. 8, pp. 1–37, 2021.

[27] Dataset Link: Available: https://www.kaggle.com/datasets/hassan06/nslkdd.

[28] S. Priyadarshini *et al.*, "Enhancing security and scalability by AI/ML workload optimization in the cloud," *Cluster Comput.*, vol. 27, no. 10, pp. 13455–13469, 2024.