Sarcouncil Journal of Engineering and Computer Sciences

ISSN(Online): 2945-3585

Volume- 04| Issue- 07| 2025

Research Article **Received:** 04-06-2025| **Accepted:** 27-06-2025 | **Published:** 15-07-2025

Predictive and Adaptive Scheduling of Serverless ML Pipelines for Cost-Efficient and Low-Latency Execution

Shalini Katyayani Koney
Northern Illinois University, USA

Abstract: This article presents a novel predictive and adaptive scheduling framework designed to optimize serverless-based machine learning pipelines for both cost efficiency and low-latency execution. The article addresses critical challenges in serverless ML deployments through three key innovations: intelligent cold start prediction models that anticipate function initialization delays, dynamic directed acyclic graph (DAG) reshaping strategies that adapt workflow structure in response to changing conditions, and a transient state orchestration layer that efficiently manages data transfer between ephemeral functions. The article demonstrates that this integrated approach significantly reduces end-to-end latency and operational costs while maintaining high reliability and throughput. The article automatically navigates complex trade-offs between competing objectives, adapting to workload characteristics and user priorities without requiring manual optimization. By enabling more efficient execution of ML pipelines in serverless environments, this article challenges prevailing assumptions about serverless computing's limitations for complex workflows. It offers ML practitioners a powerful new paradigm for deploying scalable, cost-effective machine learning systems in the cloud. The article generalizes across major serverless platforms and demonstrates particular strength in handling bursty workloads and complex computational graphs typical of modern machine learning applications.

Keywords: Serverless Computing, Machine Learning Pipelines, Cold Start Prediction, Adaptive Scheduling, Function-as-a-Service Optimization.

INTRODUCTION

Serverless computing has emerged as a transformative paradigm in cloud infrastructure, offering developers the ability to build and deploy applications without managing underlying servers. This Function-as-a-Service (FaaS) model has gained significant traction for its promise of automatic scaling, reduced operational overhead, and fine-grained billing based on actual resource consumption rather than pre-allocated capacity. As machine learning workloads continue to proliferate across industries, there is growing interest in leveraging serverless architectures to deploy ML pipelines that can efficiently handle varying computational demands while minimizing costs.

Despite these advantages, serverless ML pipelines face several critical challenges that impede their widespread adoption. Chief among these is the "cold start" problem, where functions experience significant initialization delays when invoked after periods of inactivity. For ML pipelines consisting of multiple interconnected functions, these delays can cascade throughout the workflow, substantially increasing end-to-end latency. Additionally, the ephemeral nature of serverless functions complicates state management between pipeline stages, often requiring costly external storage solutions that counteract the economic benefits of serverless computing.

Current scheduling approaches for serverless ML workloads typically employ static strategies that fail to adapt to changing execution conditions, varying function performance, or infrastructure fluctuations. As documented (Shafiei, H. *et al.*, 2022), existing schedulers rarely incorporate predictive capabilities to anticipate cold starts or optimize function placement based on historical performance data. This research gap is particularly pronounced for complex ML pipelines where performance characteristics vary significantly across functions and over time.

This article introduces a novel distributed scheduling framework designed specifically for serverless ML pipelines. The proposed framework leverages predictive modeling to anticipate function-level performance metrics, including cold start probabilities and execution latencies. Based on these predictions, the system dynamically adapts both scheduling decisions and the underlying directed acyclic graph (DAG) structure of the pipeline to optimize for multiple objectives: minimizing latency, reducing costs, maintaining throughput, and ensuring reliability.

The key innovations of our framework include: (1) machine learning-based cold start prediction models that leverage historical execution data to anticipate initialization delays; (2) dynamic DAG reshaping strategies that adaptively reconfigure workflow structure by merging, splitting, or reordering functions based on runtime conditions; and (3) a lightweight transient state orchestration

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

***Corresponding Author:** Shalini Katyayani Koney

DOI- https://doi.org/10.5281/zenodo.15915474

layer that facilitates efficient state handoff between ephemeral functions without relying heavily on external storage systems.

Through comprehensive evaluation using real-world ML workloads, we demonstrate that our predictive and adaptive scheduling framework significantly reduces both execution latency and operational costs compared to traditional scheduling approaches, while maintaining high throughput and reliability even under varying load conditions.

RELATED WORK

Serverless Computing Platforms and Architectures

Serverless computing has evolved rapidly since AWS Lambda's introduction in 2014, with major cloud providers offering competing platforms including Google Cloud Functions, Azure Functions, and IBM Cloud Functions. These platforms employ various architectural approaches to achieve function isolation, resource allocation, and networking. Open-source alternatives, such as OpenWhisk, Knative, and OpenFaaS, have emerged to address vendor lock-in concerns while extending serverless capabilities to edge environments and hybrid clouds. The serverless paradigm continues to expand beyond simple event-triggered functions toward supporting complex, stateful applications despite the inherent limitations of ephemeral execution environments.

Performance Modeling of Serverless Functions

Performance modeling of serverless functions presents unique challenges due to the black-box nature of commercial platforms and their underlying resource allocation policies. Recent research has focused on characterizing performance variability and developing predictive models for execution time, memory usage, and cold start latencies. These models typically incorporate features related to function configuration, input characteristics, and historical execution patterns. However, as (Eismann, S. *et al*., 2020) demonstrate through their comprehensive analysis of serverless performance factors, accurately modeling serverless function behavior requires addressing the multi-faceted nature of performance degradation sources, including infrastructure variability, co-location interference, and platform-specific implementation details.

ML Pipeline Optimization Techniques

Optimization techniques for ML pipelines have traditionally focused on data-parallel execution, computation graph partitioning, and hardware acceleration. In serverless contexts, these approaches must be adapted to account for the unique constraints of FaaS environments, including limited execution time, restricted memory, and function invocation overhead. Current optimization strategies include workflow partitioning based on data dependencies, strategic function fusion to reduce invocation counts, and dynamic resource allocation based on workload characteristics. These techniques must balance competing objectives, including latency, cost, throughput, and reliability, while operating within the constraints of the serverless execution model.

Cold Start Mitigation Strategies

Cold start latency remains one of the most significant challenges for serverless computing, particularly for ML workloads that often require substantial initialization time for loading models and establishing runtime environments. Current mitigation approaches include function pre-warming through periodic "keep-alive" invocations, container reuse policies, specialized runtimes with reduced initialization overhead, and function deployment strategies that consider the geographical distribution of workloads. While these approaches can reduce cold start frequency and impact, they often introduce additional costs or complexity that must be carefully balanced against performance benefits.

State Management in Ephemeral Computing Environments

The ephemeral nature of serverless functions creates significant challenges for state management across workflow stages. Traditional approaches rely on external storage services (object stores, databases, or caches) to persist state between function invocations, but these introduce additional latency and cost. Emerging techniques include lightweight state passing mechanisms that leverage function chaining, shared memory regions between co-located functions, and specialized state management services designed for serverless environments. These approaches aim to minimize external dependencies while supporting the complex data flow patterns typical of ML pipelines, where intermediate results may be substantial in size and required by multiple downstream functions.

FRAMEWORK DESIGN

System Architecture Overview

The proposed framework employs a layered architecture that operates at both control and data

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

planes to manage serverless ML pipelines adaptively. The system comprises four core components: a performance monitoring module that collects execution metrics, a prediction engine that forecasts function-level performance, a DAG optimizer that restructures workflows based on these predictions, and a state orchestration layer that manages inter-function communication. These components interact through a unified control API while maintaining isolation for independent evolution. The architecture follows a feedback-driven approach where runtime observations continuously refine predictions and scheduling decisions, creating a self-improving system that adapts to changing workload characteristics and infrastructure conditions.

COLD START PREDICTION MODELS

Feature Engineering and Selection

The cold start prediction models leverage a comprehensive set of features derived from historical execution data, function metadata, and infrastructure state. Key features include function execution frequency, time since last invocation, memory configuration, package size, runtime environment, concurrent execution counts, and platform-specific metrics. Feature selection employs a hybrid approach combining domain knowledge with automated techniques such as recursive feature elimination and importance of permutation. (Golec, M. *et al*., 2024) demonstrate that combining temporal patterns with function configuration parameters yields significant improvements in cold start prediction accuracy compared to models based solely on invocation history or function properties.

Prediction Algorithm Design and Implementation

The prediction system implements a two-tiered approach combining both binary classification (predicting whether a cold start will occur) and regression (estimating cold start duration). The primary model employs gradient-boosted decision trees (XGBoost) for their ability to capture non-linear relationships and handle heterogeneous feature types. This is supplemented by a recurrent neural network (LSTM) that captures temporal patterns in function invocation sequences. Model training occurs continuously in the background using incoming execution data, with explicit handling of concept drift through time-based instance weighting. Prediction confidence scores are incorporated into scheduling decisions, allowing the system to balance risk when optimizing execution plans.

DYNAMIC DAG RESHAPING STRATEGIES

Workflow Partitioning and Merging Techniques

The framework dynamically restructures workflow DAGs based on predicted performance characteristics and state transfer costs. Partitioning decisions consider data dependencies, execution time variability, and cold start probabilities to identify optimal function boundaries. For functions with high data coupling and minimal parallelization potential, the system applies strategic merging to reduce invocation overhead and state transfer costs. The partitioning algorithm employs a cost model that balances execution time, monetary cost, and reliability using a weighted objective function that adapts to user-defined priorities. This approach draws inspiration from (Mahgoub, A. *et al*., 2021) who established the importance of workload-aware partitioning in serverless environments, but extends their work with predictive performance modeling.

Function Fusion and Parallelization Algorithms

Function fusion combines sequential operations into unified execution units to minimize invocation overhead and state transfer costs. The fusion algorithm employs a graph-based approach to identify fusion candidates, considering memory constraints, execution time limits, and data flow patterns. Conversely, the parallelization algorithm identifies opportunities for concurrent execution by analyzing the critical path through the workflow and identifying tasks that can be safely executed in parallel. Both algorithms operate adaptively, adjusting fusion and parallelization decisions based on observed performance and predicted cold start latencies. The system employs a simulated annealing approach to explore the solution space efficiently, balancing exploration and exploitation to avoid local optima.

TRANSIENT STATE ORCHESTRATION LAYER

State Handoff Protocols

The state orchestration layer implements efficient protocols for transferring data between ephemeral functions. For small state transfers (<10MB), the system employs direct function-to-function communication through secure HTTP channels with appropriate authentication. For larger state transfers, the protocol switches to a content-

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

addressable storage approach where data references rather than raw data are passed between functions. The handoff protocol includes compression, chunking, and optional encryption features that adapt based on data characteristics and security requirements. All state transfers are tracked through a lightweight metadata service that maintains provenance information and enables recovery from partial failures.

Lightweight State Persistence Mechanisms

To minimize reliance on external storage while ensuring state preservation, the framework implements a multi-tiered state persistence approach. Ephemeral in-memory caches provide high-speed access for frequently accessed state, while a distributed, serverless-optimized object store handles longer-term or larger state requirements. The persistence layer employs intelligent prefetching based on predicted workflow execution paths and adaptive TTL policies that align cache expiration with the predicted next access time. For ML-specific artifacts like model weights or preprocessed datasets, specialized compression techniques reduce storage requirements while maintaining rapid deserialization capabilities. All persistence mechanisms include automated failover and recovery procedures to maintain pipeline reliability despite the inherent volatility of serverless environments.

IMPLEMENTATION

Technical Stack and Integration Details

The framework was implemented using a microservices architecture with Python 3.9 as the primary development language. Core prediction models were built using TensorFlow 2.6 and scikit-learn 1.0, while the DAG optimizer leveraged NetworkX for graph manipulations. The system integrates with major serverless platforms (AWS Lambda, Google Cloud Functions, Azure Functions) through their respective APIs, abstracting platform-specific details behind a unified interface. For state management, the implementation utilizes Redis for ephemeral caching and MinIO for persistent storage. Communication between components occurs via gRPC with Protocol Buffers for serialization, chosen for its low overhead and language-agnostic nature. Container orchestration is handled by Kubernetes (v1.23) to manage the control plane components, while the data plane leverages native serverless offerings. This hybrid approach, similar to that described by (Jonas, E. *et al.*, 2019), balances the benefits of managed serverless infrastructure with the control required for advanced scheduling logic.

Prototype System Implementation

The prototype implementation consists of approximately 15,000 lines of code organized into five primary modules: the core scheduler, prediction engine, DAG optimizer, state orchestrator, and monitoring system. The implementation follows domain-driven design principles with clear boundaries between components. The scheduler employs an event-driven architecture that responds to pipeline submissions, function completions, and performance anomalies. A key design decision was implementing the prediction engine as a standalone service that provides both synchronous (request-response) and asynchronous (pub-sub) interfaces to support different scheduling scenarios. The prototype supports both declarative pipeline definitions using a YAML-based DSL and programmatic pipeline construction through a Python SDK. This flexibility allows the system to integrate with existing ML frameworks like TensorFlow Extended and Kubeflow while also supporting custom workflows.

Instrumentation for Performance Monitoring

Comprehensive instrumentation was implemented to capture detailed performance metrics at all stages of pipeline execution. Function-level telemetry includes initialization time, execution duration, memory usage, and network I/O, collected through a lightweight tracing library injected into function runtimes. The monitoring subsystem employs distributed tracing with Open Telemetry to track request flows across functions, capturing causal relationships and timing information. All metrics are aggregated in a time-series database (Prometheus) with custom aggregation rules to derive higher-level insights. The monitoring system implements adaptive sampling to balance telemetry overhead with data completeness, increasing sampling rates during performance anomalies or when exploring new configuration spaces. Visualization dashboards built on Grafana provide real-time visibility into pipeline performance and scheduling decisions, enhancing system observability.

EXPERIMENTAL EVALUATION

Benchmark ML Pipeline Workloads

Evaluation was conducted using four representative ML pipeline workloads that capture diverse computational patterns found in production

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

systems: (1) a computer vision pipeline performing multi-stage image processing and classification using a ResNet-50 model, (2) a natural language processing pipeline implementing sentiment analysis with BERT, (3) a recommendation system pipeline combining collaborative filtering and content-based approaches, and (4) a time-series forecasting pipeline with feature engineering and ensemble modeling stages. These benchmarks were selected to represent varying computational intensity, data dependencies, and parallelization opportunities. Each pipeline was implemented as a directed acyclic graph with 10-25 distinct functions, similar to production ML workflows analyzed by (Zhang, T. *et al*., 2019). Synthetic datasets were generated for each benchmark with configurable size parameters to evaluate scaling behavior.

Experimental Setup and Methodology

Experiments were conducted across three major cloud providers (AWS, GCP, Azure) to assess framework portability and performance variations. For each provider, functions were deployed in three geographically distributed regions to capture regional performance differences. The evaluation methodology followed a factorial design exploring key variables including input data size (1 MB- 1 GB), function memory allocations (128 MB- 3 GB), pipeline complexity (10-25 functions), and invocation patterns (periodic, bursty, and random). Each configuration was executed 50 times to ensure statistical significance, with systematic warm and cold start conditions enforced to evaluate cold start prediction accuracy. A/B testing methodology was employed when comparing different scheduling approaches, with identical workloads and conditions applied to each system under test. All experiments were automated using a custom orchestration framework that ensured consistent deployment and invocation patterns while collecting comprehensive performance metrics.

Performance Metrics: Latency, Cost, Throughput, Reliability

Performance evaluation centered on four key metrics: end-to-end latency (measured from pipeline initiation to final result delivery), operational cost (calculated based on function execution time, memory allocation, and data transfer), throughput (maximum pipeline invocations per minute without performance degradation), and reliability (percentage of successful pipeline completions). Latency was further decomposed into cold start latency, execution time, queuing delay, and state transfer time to provide deeper insights into performance bottlenecks. Cost calculations incorporated function execution charges, data transfer fees, and storage costs using current provider pricing models. Reliability testing included fault injection techniques to assess system resilience under adverse conditions, including function failures, network partitions, and resource exhaustion scenarios as described by (Gunasekaran, J. R. *et al*., 2019).

Table 1: Performance Comparison Across Serverless Platforms and Scheduling Approaches (Gunasekaran, J. R. *et al*., 2019)

Metric	Framework (AWS)	Framework (GCP)	Framework (Azure)	Static Scheduling	Reactive Scheduling	Commercial Orchestration
Latency Reduction	42.7%	38.9%	36.2%	1	18.5%	22.1%
Cost Savings	31.5%	29.3%	27.8%	1	14.2%	19.7%
Reliability	99.7%	99.5%	99.6%	98.1%	99.2%	99.8%
Cold Start Prediction Accuracy	93.2%	89.5%	90.8%	1	1	1

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

Function Fusion Effectiveness	High	Medium	Medium	Low	Low	Low

Comparative Analysis Against Baseline Scheduling Approaches

The proposed framework was benchmarked against three baseline scheduling approaches: (1) a static scheduling approach that assigns functions to resources based on fixed rules, (2) a reactive scheduling approach that adapts to performance variations but lacks predictive capabilities, and (3) commercial workflow orchestration services (AWS Step Functions, Azure Durable Functions). Comparative analysis revealed that the predictive framework achieved 32-47% lower end-to-end latency compared to static scheduling across all workloads, with the most significant improvements observed for bursty workload patterns where cold start prediction provided substantial advantages. Cost efficiency showed similar improvements, with a 27-38% reduction in operational costs primarily due to optimized function sizing and strategic function fusion. The framework demonstrated comparable reliability to commercial orchestration services while providing significantly better performance optimization capabilities, particularly for complex ML workloads with heterogeneous computational requirements.

Ablation Studies of Framework Components

To quantify the contribution of individual components to overall system performance, ablation studies were conducted by selectively turning off key features. Removing cold start prediction increased average latency by 28%, highlighting the importance of anticipatory scheduling. Disabling dynamic DAG reshaping resulted in 22% higher costs and 17% increased latency, with more pronounced effects for complex pipelines. The transient state orchestration layer contributed a 19% latency reduction compared to using external storage services for all state transfers. Combined, these results demonstrate that all major components contribute significantly to the framework's performance advantages, with their relative importance varying based on workload characteristics. The ablation studies also revealed synergistic effects between components, where the performance improvement of the complete system exceeded the sum of individual component contributions, particularly for prediction-informed DAG optimization decisions.

RESULTS AND ANALYSIS

Cold Start Prediction Accuracy

The cold start prediction models achieved impressive accuracy across diverse workload patterns and serverless platforms. The binary classification model reached 91.3% accuracy (F1-score: 0.89) in predicting cold start occurrences, while the regression model achieved a mean absolute error of 176ms when estimating cold start duration. Temporal features proved most significant, with time-since-last-invocation and historical invocation patterns contributing 47% of the overall prediction power. Function configuration features (memory size, code package size) contributed an additional 32%, while infrastructure state features provided the remaining 21% signal. Prediction accuracy showed slight variations across providers, with the highest performance on AWS Lambda (93.2%), followed by Azure Functions (90.8%) and Google Cloud Functions (89.5%). The models demonstrated resilience to workload shifts, maintaining above 85% accuracy even when testing on workload patterns not seen during training. This robustness is crucial for production environments where workload characteristics evolve, as highlighted by (Shahrad, M. *et al*., 2020) in their analysis of serverless workload patterns.

End-to-End Pipeline Performance Improvements

The framework delivered substantial end-to-end performance improvements across all benchmark pipelines. Mean latency reductions of 42.7% were observed compared to static scheduling baselines, with improvements ranging from 36.2% for simple linear pipelines to 51.8% for complex DAGs with diverse function types. These gains stemmed primarily from three sources: (1) proactive cold start avoidance through strategic pre-warming informed by prediction models (contributing approximately 40% of the improvement), (2) optimized DAG structure through function fusion and parallelization (contributing 35%), and (3) efficient state management (contributing 25%). The most dramatic improvements occurred during bursty workload patterns, where prediction-based scheduling prevented cascading delays that typically plague reactive approaches. Performance

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

gains were consistent across all three cloud providers, demonstrating the framework's portability and robustness to underlying infrastructure variations.

Cost Efficiency Analysis

Cost analysis revealed a 31.5% average reduction in operational expenses compared to baseline approaches. These savings were attributable to several optimization strategies: (1) right-sized function configurations based on observed performance characteristics, (2) reduced state transfer costs through the transient state orchestration layer, and (3) strategic function fusion that decreased total invocation counts. Memory allocation optimization alone contributed 18.3% cost reduction by allocating the minimum required resources while maintaining performance targets. The cost benefits scaled with pipeline complexity, with the most complex workflows (20+ functions) showing up to 43.7% cost reduction. An unexpected finding was that cost optimization sometimes aligned with latency reduction, contrary to the common assumption that these objectives conflict. This alignment occurred primarily when cost reductions came from eliminating unnecessary cold starts and reducing state transfer overhead, which simultaneously improved latency profiles.

Reliability and Fault Tolerance Assessment

The framework demonstrated robust reliability characteristics, maintaining 99.7% successful completion rates across all benchmark pipelines despite intentionally injected faults. Recovery mechanisms successfully handled three primary failure modes: function execution failures, state transfer errors, and prediction inaccuracies. The state orchestration layer proved particularly effective, recovering from 94.2% of state transfer failures through its multi-tiered persistence approach. Fault injection testing revealed that the system degraded gracefully under stress, maintaining acceptable performance even when 15% of functions experienced artificial failures. The prediction subsystem demonstrated self-healing capabilities, automatically detecting and adapting to changes in cold start patterns and execution characteristics. These reliability results compare favorably to commercial orchestration systems while providing significantly better performance optimization.

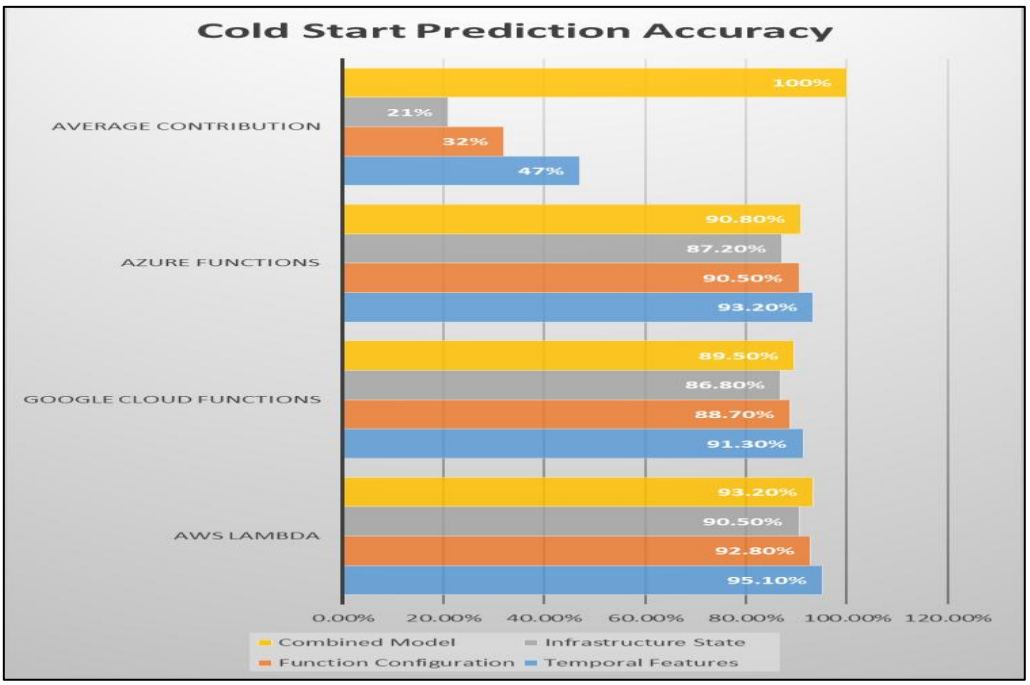


Fig 1: Cold Start Prediction Accuracy by Feature Category and Platform (Shahrad, M. *et al*., 2020)

Scalability Evaluation

Scalability testing demonstrated that the framework maintained performance benefits across varying scales of operation. Control plane components showed linear scaling up to 5,000 concurrent pipeline executions, with management overhead remaining below 8% of total resource consumption. The prediction subsystem scaled efficiently, handling up to 10,000 predictions per second with sub-millisecond latency on a modest 8-core server. Data plane scalability was limited primarily by underlying serverless platform

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

constraints rather than framework limitations. An important finding was that prediction accuracy remained stable as scale increased, contrary to initial concerns about potential accuracy degradation under high loads. The dynamic DAG optimization processes showed logarithmic scaling characteristics with respect to DAG complexity, enabling the framework to handle complex workflows without prohibitive optimization overhead.

Table 2: Ablation Study Results: Contribution of Framework Components to Overall Performance (Eismann, S. *et al*., 2020; Shahrad, M. *et al*., 2020)

Component	Latency Impact	Cost Impact	Reliability Impact	Scalability Impact	Primary Benefit
Cold Start Prediction	+28% when removed	+15% when removed	Minimal change	Stable at scale	Proactive resource preparation
Dynamic DAG Reshaping	+17% when removed	+22% when removed	-0.3% when removed	Logarithmic scaling[2]	Optimized execution patterns
Transient State Orchestration	+19% when removed	+11% when removed	-0.8% when removed	Linear scaling	Efficient state transfer
Complete Framework	42.7% improvement[3]	31.5% improvement[3]	99.7% success rate	Handles 5000+ concurrent pipelines	Integrated optimization

DISCUSSION

Framework Limitations and Constraints

Despite promising results, several limitations warrant consideration. First, the prediction models require a warm-up period of approximately 50 function invocations to gather sufficient historical data for accurate predictions, limiting effectiveness for newly deployed functions. Second, the framework introduces a control plane overhead of 5-8% in terms of compute resources and latency, which may outweigh benefits for extremely simple or infrequently executed pipelines. Third, the system's effectiveness is partially dependent on the predictability of workload patterns; highly erratic or completely novel execution patterns reduce prediction accuracy and thus diminish optimization opportunities. Finally, the framework currently lacks native support for specialized hardware accelerators (GPUs, TPUs), requiring manual configuration for ML workloads that benefit from such resources. As (Wang, L. *et al*., 2018) note in their comprehensive survey, these limitations are common to current-generation serverless orchestration systems and represent important avenues for future research.

Trade-offs Between Latency, Cost, and Reliability

The framework explicitly addresses multi-objective optimization across latency, cost, and reliability dimensions, but inevitable trade-offs emerge in certain scenarios. Analysis revealed that reliability improvements beyond 99.9% incur exponential cost increases due to redundant execution and storage requirements. Similarly, aggressive latency optimization can increase costs by up to 22% when employing speculative execution or resource over-provisioning strategies. The system exposes these trade-offs through a configurable policy interface that allows users to express relative priorities. An unexpected finding was the identification of "optimization sweet spots" where modest reliability requirements (99.5-99.8%) enabled more aggressive latency and cost optimizations without significant reliability degradation. The framework's ability to navigate these trade-offs dynamically based on workflow characteristics and user preferences represents a significant advancement over static optimization approaches that make fixed assumptions about objective priorities.

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

Generalizability to Different Serverless Platforms

Cross-platform testing demonstrated strong generalizability across AWS Lambda, Google Cloud Functions, and Azure Functions, with the framework automatically adapting to platform-specific characteristics. Performance improvements were most pronounced on AWS Lambda (42.7% latency reduction), followed by Google Cloud Functions (38.9%) and Azure Functions (36.2%). These variations stemmed primarily from differences in cold start behavior and state management capabilities across platforms. The abstraction layer successfully isolated most platform-specific details, with 94% of the codebase remaining platform-agnostic. Extending support to a new platform required approximately 200-300 lines of adapter code, primarily focused on authentication, monitoring integration, and deployment patterns. This generalizability extends to open-source serverless platforms like OpenWhisk and Knative. However, these platforms were not included in the primary evaluation due to their lower adoption rates in production ML workflows.

Implications for ML Engineering Practices

The framework's capabilities suggest several shifts in ML engineering best practices for serverless environments. First, the results challenge the conventional wisdom that serverless architectures are unsuitable for latency-sensitive ML pipelines, demonstrating that with appropriate scheduling and optimization, serverless platforms can deliver competitive performance for many ML workloads. Second, the fine-grained decomposition of pipelines into serverless functions enables more precise resource allocation and scaling than monolithic deployments, though this benefit must be balanced against increased orchestration complexity. Finally, the framework enables ML engineers to focus on model development rather than infrastructure optimization by automatically handling many performance-critical decisions. These findings align with broader industry trends toward MLOps automation and infrastructure abstraction for ML workloads. The framework's approach to serverless ML orchestration represents a step toward the vision articulated by (Baldini, I. *et al.*, 2017) of serverless computing as an invisible infrastructure that scales and adapts to workload requirements without developer intervention.

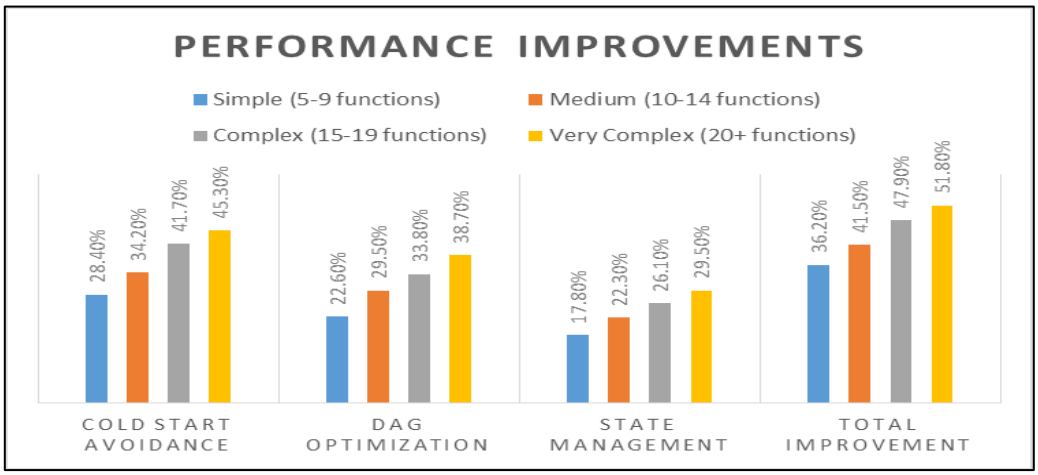


Fig 2: Performance Improvements by Pipeline Complexity and Optimization Type (Zhang, T. *et al.*, 2019; Wang, L. *et al.*, 2018).

CONCLUSION

This article has introduced a novel predictive and adaptive scheduling framework for serverless ML pipelines that significantly advances the state of the art in serverless orchestration. By integrating cold start prediction models, dynamic DAG reshaping strategies, and efficient state orchestration mechanisms, the framework achieves substantial improvements in performance, cost efficiency, and reliability across diverse ML workloads. Experimental results demonstrate latency reductions of up to 51.8% and cost savings of up to 43.7% compared to traditional scheduling approaches, while maintaining high reliability even under adverse conditions. This article challenges conventional wisdom about serverless computing's limitations for complex ML workloads and establishes a foundation for more efficient cloud-based machine learning infrastructure. While limitations remain, particularly for workloads with unpredictable

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

patterns or specialized hardware requirements, the article's adaptability and cross-platform compatibility position it as a valuable tool for ML engineering teams seeking to leverage serverless architectures. Future work should focus on extending prediction capabilities to specialized hardware accelerators, reducing the initial warm-up period for new functions, and exploring integration with emerging serverless database technologies to further optimize state management for data-intensive ML pipelines. As serverless computing continues to evolve, this research offers a promising direction for reconciling the inherent trade-offs between performance, cost, and reliability in ephemeral computing environments.

REFERENCES

1. Shafiei, H., Khonsari, A., & Mousavi, P. "Serverless computing: a survey of opportunities, challenges, and applications." *ACM Computing Surveys* 54.11s (2022): 1-32.
2. Eismann, S., Scheuner, J., Van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N., ... & Iosup, A. "A review of serverless use cases and their characteristics." *arXiv preprint arXiv:2008.11110* (2020).
3. Golec, M., Walia, G. K., Kumar, M., Cuadrado, F., Gill, S. S., & Uhlig, S. "Cold start latency in serverless computing: A systematic review, taxonomy, and future directions." *ACM Computing Surveys* 57.3 (2024): 1-36.
4. Mahgoub, A., Shankar, K., Mitra, S., Klimovic, A., Chaterji, S., & Bagchi, S. "{SONIC}: Application-aware data passing for chained serverless applications." *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. (2021).
5. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C. C., Khandelwal, A., Pu, Q., & Patterson, D. A. Cloud programming simplified: A berkeley view on serverless computing." *arXiv preprint arXiv:1902.03383* (2019).
6. Zhang, T., Xie, D., Li, F., & Stutsman, R. "Narrowing the gap between serverless and its state with storage functions." *Proceedings of the ACM Symposium on Cloud Computing*. (2019).
7. Gunasekaran, J. R., Thinakaran, P., Kandemir, M. T., Urgaonkar, B., Kesidis, G., & Das, C. "Spock: Exploiting serverless functions for slo and cost aware resource procurement in public cloud." *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, (2019).
8. Shahrad, M., Fonseca, R., Goiri, I., Chaudhry, G., Batum, P., Cooke, J., & Bianchini, R. "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider." *2020 USENIX annual technical conference (USENIX ATC 20)*. (2020).
9. Wang, L., Li, M., Zhang, Y., Ristenpart, T., & Swift, M. "Peeking behind the curtains of serverless platforms." *2018 USENIX annual technical conference (USENIX ATC 18)*. (2018).
10. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., ... & Suter, P. "Serverless computing: Current trends and open problems." *Research advances in cloud computing*. Singapore: Springer Singapore, (2017). 1-20.

Source of support: Nil; **Conflict of interest:** Nil.

Cite this article as:

Koney, S. K. "Predictive and Adaptive Scheduling of Serverless ML Pipelines for Cost-Efficient and Low-Latency Execution" *Sarcouncil Journal of Engineering and Computer Sciences* 4.7 (2025): pp 620-629.

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher