Configuring Dynamic Multi-Stage Serverless Pipelines for Video Processing with Minimal Profiling Overhead

Jiaye Zhang
Tsinghua University
Beijing, China
jy-zhang21@mails.tsinghua.edu.cn

Hongyi Wang
Tsinghua University
Beijing, China
hongyi-w21@mails.tsinghua.edu.cn

Peiru Yang
Tsinghua University
Beijing, China
ypr21@mails.tsinghua.edu.cn

Zili Meng*
Hong Kong University of Science and Technology
Hong Kong, Hong Kong
zilim@ust.hk

Mingwei Xu*
Tsinghua University
Beijing, China
xumw@tsinghua.edu.cn

Abstract

Serverless computing has become a promising paradigm for video processing workflows, offering simplified deployment and flexible management of business logic. However, the dynamic, multi-stage nature of video processing pipelines poses significant challenges for traditional serverless resource management, particularly in efficiently modeling optimal configurations and adapting to rapidly evolving pipeline structures. To address this challenge, we propose ConfigNavigator, a video pipeline resource tuning framework capable of adapting to dynamic inputs and pipeline structures with minimal overhead. In the offline phase, ConfigNavigator models function execution time distributions at the fundamental operation level and leverages graph theory to decompose complex video processing pipelines, thereby obtaining optimal configurations with minimal overhead. In the online phase, it dynamically adjusts function configurations on critical paths through real-time performance feedback, ensuring pipeline performance stability across varying workloads. We evaluate ConfigNavigator using real video streams on the commercial serverless platform AWS Lambda. Compared to state-of-the-art baselines, ConfigNavigator reduces configuration search time by 94.11% while decreasing end-to-end pipeline processing time by 13.97%.

CCS Concepts

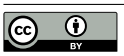
• **Computing methodologies** → **Parallel computing methodologies**; • **Computer systems organization** → **Cloud computing**; • **Information systems** → **Multimedia streaming**.

Keywords

serverless; video processing

ACM Reference Format:
Jiaye Zhang, Hongyi Wang, Peiru Yang, Zili Meng, and Mingwei Xu. 2025. Configuring Dynamic Multi-Stage Serverless Pipelines for Video Processing with Minimal Profiling Overhead. In *Proceedings of the 33rd ACM International Conference on Multimedia (MM '25), October 27–31, 2025, Dublin, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3746027.3755046

*Corresponding authors.



MM '25, Dublin, Ireland

ACM ISBN 979-8-4007-2035-2/2025/10
https://doi.org/10.1145/3746027.3755046

1 Introduction

Video traffic, accounting for approximately one-third of downstream traffic [36], has become an inseparable part of our daily lives. With the popularity of User-Generated Content [48] and video social applications, video processing workflows have evolved beyond traditional encoding and decoding to incorporate object detection, beautification, super-resolution, and other technologies [51], resulting in multi-stage, dynamic processing pipelines [25]. Decoupling processing logic from deployment specifications to improve maintainability and scalability has emerged as a significant trend for video processing workflows to address increasingly complex inter-service processing logic and conflicts in business code management across teams [11, 21]. Serverless computing abstracts infrastructure management away from application development [10, 37]. By delegating infrastructure management to cloud service providers, development teams can focus on business logic while leveraging elastic scaling and fine-grained billing features that enable users to achieve optimal resource utilization through flexible configuration options for their diverse computational needs [26].

Resource tuning strategies have been proposed to effectively harness serverless computing for pipeline application to meet Service Level Objective(SLO) requirements as shown in Table 1, which can be broadly categorized into two types: **Resource tuning based on expert knowledge** [19, 24, 30, 34, 35, 47] relies on domain-specific insights about the pipeline and serverless, which may fail if runtime conditions deviate from initial assumptions(such as constraint violations or intermediate optimization parameter mismatches). **Resource tuning based on learning** [2, 12, 20, 31, 32, 45, 52, 53] adopts data-driven, end-to-end optimization approach, bypassing explicit pipeline modeling or queueing theory analysis. It directly learns and optimizes the pipeline's overall performance under certain assumptions which can be deviated(e.g., Gaussian runtime distribution) [2], leading to ineffective and sometimes completely erroneous search results.

However, there remains a significant gap in applying these approaches to video processing scenarios:

Table 1: Comparison of existing pipeline resource management systems with ConfigNavigator based on whether they (a) have low overhead(both in measuring and infrastructure), (b) address the challenges of meeting performance targets for general video pipelines, and (c) can adapt to dynamic workflow and changing pipeline. ●◐○denotes the level of overhead, where ●indicates high overhead, ◐indicates moderate overhead, and ○indicates low overhead.

System	Year	Tuning	Challenges			
			Overhead		Accurate & Fast tuning	Adaptability
			Measurement	System		
COSE [2]	2020	Bayesian	◐	○	✓	×
Charmseeker [52]	2022	Sequential Bayesian	◐	○	✓	×
Aquatope [53]	2022	Batched Bayesian	◐	○	✓	×
Llama [34]	2021	Heuristic	◐	○	Limited	✓
SLAM [35]	2022	Heuristic	●	○	Limited	×
FaaSConf [45]	2024	Reinforcement Learning	●	◐	✓	×
SIMPPO [31]	2022	Reinforcement Learning	●	◐	✓	×
AWARE [32]	2023	Reinforcement Learning	●	◐	✓	✓
ConfigNavigator	-	Prediction & Heuristic	○	○	✓	✓

High Measurement Overhead in Serverless Analytics: In video processing workflows, the multi-pipeline nature causes inaccurate performance characterizations to create cascading effects that further exacerbate performance degradation [49]. However, accurate performance characterization of serverless functions requires repeated executions to mitigate cold-start variability and multi-tenancy interference [27, 41, 44]. Achieving representative performance distributions (capturing central tendencies and noise characteristics) with minimal measurement overhead remains a critical yet unresolved challenge in video processing workflow optimization.

High-dimensional Configuration Space of Pipelines: Due to the need to support the rich variety of video processing tasks available today, the length and complexity of pipelines create a vast and intricate configuration space with prevalent local optima [28, 29, 33]. Evaluating each configuration incurs significant time and resource costs, making brute-force search approaches impractical. Existing methods face fundamental trade-offs between efficiency and quality: knowledge-based approaches lack scalability in high-dimensional spaces, while learning-based techniques frequently converge to suboptimal configurations.

Dynamic Pipeline Updates: Video pipelines face inherent runtime dynamics from content changes (resolution, encoding, motion complexity) and structural mutations, resulting in non-stationary execution paths and latency distributions [17, 18]. Current static optimization paradigms [7, 34, 46] fail to generalize across this variability spectrum—either focusing too narrowly on input features while neglecting pipeline reconfiguration events, or requiring impractical retraining overhead when encountering unseen scenarios.

Therefore, we propose ConfigNavigator, a dynamism-oriented video pipeline resource tuning framework that enables low-overhead performance prediction through basic operation modeling to achieve global optimization:

Lightweight performance modeling with basic operations: ConfigNavigator enables lightweight function profiling through a decomposition-recomposition method. We observe that all functions are composed of basic operations (e.g., arithmetic operations, I/O calls), whose execution time distributions exhibit stable characteristics across serverless platform configurations. This stability allows their relationship with configurations to be accurately modeled by a neural network trained on a small-scale dataset which includes offline-collected basic operation benchmarks and online-measured timing metrics. We then reconstruct a function's execution time by combining profiles of its involved basic operations instead of directly measuring the execution time, thus eliminating the need for prolonged measurement.

Hierarchical dimensionality reduction strategy: By leveraging the inherent parallelism of video pipelines, ConfigNavigator groups parallel nodes to reduce the complexity of search space. Within each group, a Bayesian-Exhaustive hybrid search is performed to identify optimal configuration under given constraints. These parallel nodes are then abstracted into equivalent nodes, further simplifying the pipeline structure to reducing search complexity.

Online adaptive controller: ConfigNavigator continuously monitors the real-time performance of the pipeline and analyzes its critical path. ConfigNavigator then dynamically adjusts resource allocation—either constricting resources on non-critical paths or relaxing them on critical paths, ensuring robust performance despite variations in input stream characteristics and pipeline structures.

Our contributions can be summarized as follows:

(1) Through extensive measurements of 50 popular videos collected from Bilibili and composite pipelines combined from 7 representative video processing functions, we summarize the key factors affecting serverless pipeline tuning from three dimensions: measurement overhead, configuration space, and dynamic updates.

(2) We propose a multi-stage tuning framework, ConfigNavigator, which combines offline and online phases to search the entire configuration space with minimal overhead, while effectively adjusting critical paths in response to dynamic input videos and changes in pipeline structure.

(3) We implemented ConfigNavigator on AWS Lambda [5] and evaluated it in multi-step video analysis pipelines. ConfigNavigator significantly outperforms existing learning-based methods across multiple metrics. It achieves over 94.11% reduction in configuration search time. When processing dynamic video inputs, it reduces end-to-end latency by 13.97% across applications in average.

2 Background and Motivation

2.1 Serverless Video Processing

Modern video processing pipelines consist of multiple heterogeneous stages, such as decoding, filtering, object detection, and encoding, each exhibiting distinct computational characteristics, resource utilization patterns, and execution time distributions [34]. These pipelines must ensure robust SLO guarantees despite highly dynamic conditions, including unpredictable variations in frame rates (ranging from 30 to 120 fps), resolution shifts (from HD to 4K), and fluctuating scene complexity [18, 22, 23]. To address these challenges, prior studies have explored approaches such as adaptly adjusting configurations such as neural network model sizes [15] and orchestrating pipeline functions to meet SLO requirements [6, 14].

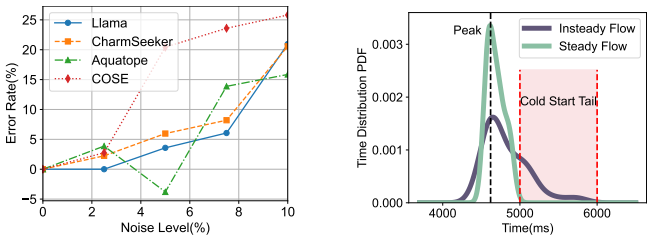
Among these efforts, serverless computing has emerged as a promising paradigm for scalable and cost-efficient video processing by offloading infrastructure management to cloud providers, leaving users responsible only for configuring pipeline resources. Expert knowledge-based resource management methods can achieve computational overhead as low as 0.01 milliseconds [34, 35], but face suboptimal solutions due to environmental noise and distortion of intermediate variables under unpredictable circumstances. Bayesian optimization methods (such as COSE [2], Charmseeker [52], and Aquatope [53]) enhance the ability to handle high-dimensional search spaces by improving noise processing mechanisms and pipeline segmentation strategies. However, two fundamental challenges remain: limited transferability when functions change, necessitating retraining of the entire model; and difficulty in modeling serverless platform noise, affecting proxy function effectiveness.

Existing methods struggle with limited sampling. While reinforcement learning incurs low inference overhead (milliseconds), it poses practical issues in serverless settings: additional dependencies increase storage requirements by 200-500MB, leading to increased cold start latency; data collection is also costly—FaaSConf [45] requires 31,000 samples over 43 days, and SIMPPO [31] needs 240ms of environment interaction per iteration. These limitations constitute practical barriers in production environments requiring rapid deployment and adaptation.

2.2 Measurement and Challenges

We perform characterization study based on a diverse dataset comprising 50 popular music videos from Bilibili's [8] trending section (collected in November 13, 2024), processed through 7 representative video processing function services for 100 times. These services encompass CPU-intensive operations, network-intensive tasks, and ML inference workloads, which are randomly combined to construct pipelines with varying architectures. The experimental setup deploys these composite pipelines on AWS Lambda [5], while the input streaming patterns are simulated using traces from Microsoft Azure [38] to reflect realistic workload scenarios. Below, we highlight the key challenges identified through our measurements.

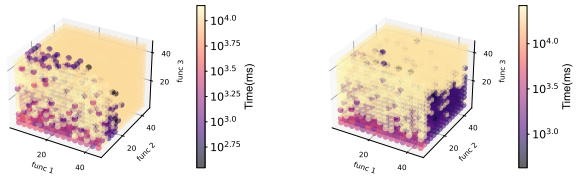
Measurement overhead with cold start: Accurate measurement is critical for pipeline configuration, as small inaccuracies in the measurement propagate into larger biases, which can fundamentally undermine the reliability and applicability of prior algorithmic assumptions, as shown in Figure 1a.



(a) Noise-induced measurement and its impact in face detection pipeline.

(b) Time distribution of steady and insteady measurement, showing cold start impact.

Figure 1: Measurements of noise effect and characteristics.

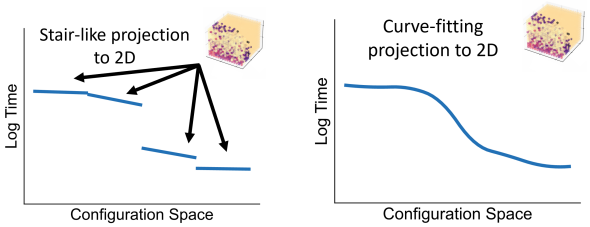


(a) 3-function pipeline (branch).

(b) 3-function pipeline (sequence).

Figure 2: 3 function pipeline's configuration space and end-to-end latency with exceed-bound penalty. The x, y and z axis denote the memory configuration of each function.

In serverless performance profiling, the accuracy of measurements is inherently challenging due to the transient and dynamic nature of cloud execution environments. Cold starts are a primary factor distorting measurement validity, causing unpredictable latency spikes—up to 213.08% in our measurements—due to factors like non-deterministic sandbox setup and logging initialization (Figure 1b). Moreover, the transient nature of cold start necessitates a large number of repeated invocations to statistically filter out such noise, rendering traditional single-run benchmarking approaches ineffective for serverless environments.



(a) Staired configuration graph.

(b) Continuous configuration graph.

Figure 3: End-to-end latency under different configurations, which exhibits a stair-like pattern.

Large configuration space and Performance stair: Pipeline operations offer a variety of knobs that can be tuned to improve latency and resource use. For example, serverless pipelines expose multiple tunable parameters per function, including memory provisioning (e.g., 128MB-10GB), and concurrency limits (e.g., 1-1000 instances), creating a high-dimensional configuration space(e.g., over 1 million possible configurations in a simple 4-function pipeline face-detection) that grows exponentially with pipeline length. Determining configurations is challenging due to the combinatorial

explosion of the configuration space driven by the number of tunable parameters, pipeline stages. Our measurement reveals that optimal configurations form sparse, localized clusters in the parameter space (Figure 2), where performance remains stable within regions but exhibits abrupt transitions (14.75–38.02% latency jumps) between adjacent near-optimal zones (Figure 3). This stair-like performance landscape creates a fundamental tension: optimization strategies must efficiently explore globally to identify these isolated high-performance regions while exploiting locally to refine configurations within plateaued neighborhoods—all while overcoming measurement noise and combinatorial search complexity.

Dynamic pipeline update: The serverless video processing pipeline must support frequent, low-latency reconfiguration to accommodate the rapid iteration of filters and video processing algorithms. An example is TikTok's AR ecosystem [43], where users continuously upload custom filters—necessitating immediate pipeline adaptation while maintaining strict latency guarantees. Furthermore, the structure of pipelines can change dynamically based on input content. For example, in face detection tasks, the number of processing branches may vary depending on the number of detected faces. While this adaptability is essential for handling diverse inputs, it complicates pipeline design, particularly when maintaining low latency and high throughput under variable workloads.

3 ConfigNavigator Design

We propose an end-to-end, multi-stage resource tuning framework as illustrated in Figure 4: a *pre-trained prediction model* (Sec 3.1) that captures the impact of resource configurations on function latency for low-overhead profiling, an offline search that *segments the pipeline* to reduce complexity (Sec 3.2), and an online tuner that searches within the neighborhood of optimal configurations at *adaptive granularity* (Sec 3.3).

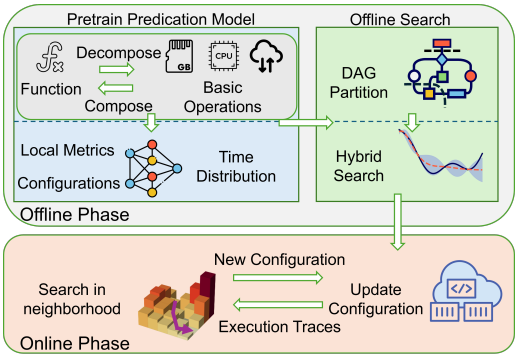


Figure 4: Framework of ConfigNavigator.

3.1 Performance Modeling

ConfigNavigator predicts the time distribution of unseen functions using low-level detailed profiling through pretrained models. We model both the central tendency and extreme tail behavior of the data to fit the distribution curve of function execution times, enabling ConfigNavigator to accurately model the performance of unseen functions. When functions are newly deployed in a pipeline, performance predictions can be made using locally profiled basic operations from a single run to determine resource allocation.

Function set synthesized from basic operations: Selecting representative functions for testing is crucial for ensuring the reliability and universality of measurements. Inspired by the Sizeless [13] approach, we generate test functions by combining modules, each representing a key operation in serverless computing. These modules include floating-point computation, local cache read/write, remote storage read/write, and network communication. By combining these modules, we generate 400 diverse functions designed to simulate common serverless workloads such as data processing, storage-intensive tasks, and machine learning inference, ensuring comprehensive coverage of typical serverless application scenarios. Subsequently, we deploy these functions in a local Docker environment and profile the basic operations listed in Table 2.

Test Setup on Serverless Platforms: We measure execution times and resource consumption metrics for 400 different functions across 32 memory configurations, ranging from 128MB to 4096MB in 128MB increments. These configurations cover all available memory sizes on AWS Lambda, capturing both resource-constrained and resource-rich execution environments. To minimize the effects of cold starts and "traffic tide" fluctuations (irregular changes due to multi-tenant competition [39]), we invoke each function every five minutes, ensuring they remain in a warm state during testing [38, 40]. By conducting long-term testing over two weeks, we ensure our measurements captured various resource contention scenarios, providing a robust dataset for subsequent analysis.

Time Distribution Modeling: To model the performance behavior of serverless functions, we develop two independent three-layer neural network models. The first network predicts average execution time and 95% tail latency, capturing central and extreme performance metrics. The second network estimates the variance of execution times, thus modeling time distribution under different conditions. Each network consists of three fully connected layers with 64, 32, and 16 neurons respectively. This architecture is chosen based on empirical evaluation, balancing model complexity and prediction accuracy.

Based on the predicted time parameters, we construct two distinct probability distributions: a normal distribution fitted using mean latency and variance, capturing the central tendency and variability of the data; and a long-tailed Pareto distribution fitted with mean latency and 95% tail latency, modeling the extreme tail behavior observed in real-world scenarios. To combine these two distributions, we introduce a smoothing function between the tail of the normal distribution and the body of the Pareto distribution, ensuring seamless fusion between the two distributions while preserving the characteristics of both the central region and the tail.

Table 2: Methods and their corresponding metrics. Memory snapshots is a sequence including the time, size and number of memory allocation during execution.

Method	Operation Metric
perf	Instructions, Branches, Context-switches
valgrind	Memory snapshots, Heap size
/proc	Stack size
wrapper	Bytes sent, Bytes received, Packets sent, Packets received
strace	I/O throughput

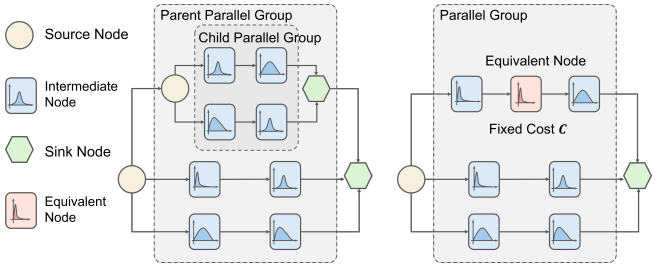


Figure 5: The procedure of Partition and Substitution.

3.2 Cost-based Offline Pipeline Tuning

We initially model the pipeline as a Directed Acyclic Graph(DAG) $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ represents the set of tasks and $E \subseteq V \times V$ represents the dependencies between tasks. In this model, each task $v_i \in V$ requires a certain amount of computational resources r_i and has an execution time $t_i(r_i)$, which depends on the resource allocation. The total execution time of the pipeline is denoted as $f(r)$, which depends on the resource allocation across all tasks. The optimization objective is:

$$\underset{r}{\text{minimize}} \quad f(r) \quad \text{s.t.} \quad c \leq C_{\text{bound}},$$

where $C_{\text{bound}}$is the cost constraint for the pipeline.

3.2.1 Partitioning based on Parallel Structure. The optimal task configuration problem, which involves determining the most efficient allocation of tasks under given constraints, is a NP-complete problem [42]. In serverless pipeline configuring, the execution time of each module is not fixed due to varying resource configurations. To address the computational intractability of the task configuration problem in serverless pipelines, we adopt a divide-and-conquer strategy. By decomposing the pipeline into smaller subsets of tasks, we effectively reduce the problem's scale, enabling more efficient analysis and optimization. Since branches in a parallel group originate from the same source node and converge at the same sink node, they ideally share a common execution deadline to prevent straggler effects [50] and ensure balanced execution. Thus, a critical challenge in decomposing the large configuration space is how to efficiently model and simplify these parallel paths while preserving the overall pipeline structure and performance constraints.

Node Classification: We classify all nodes in the DAG into three categories: source nodes, sink nodes, and intermediate nodes. Source nodes are nodes with out-degree bigger than 1, sink nodes are nodes with in-degree bigger than 1. Intermediate nodes are other nodes that neither qualify as source nodes nor sink nodes.

Parallel Group Construction: For each source node, we identify all branch paths originating from this node and check all their sink nodes. Count the occurrences of each sink nodes, and retain only those appearing in at least two distinct paths. Branches sharing the same source node and sink node are grouped into a parallel group, denoted as $\{G_1, G_2, \ldots, G_m\}$, where the nodes within each G_m exclude the source node and sink node.

Check Parallel Group: A parallel group G_m is defined as a child parallel group of G_n if the following condition holds: for any node v_i, if $v_i \in G_m$, then v_i must belong to G_n. Conversely, G_n is referred to as the parent parallel group of G_m. When a parent parallel group is composed of multiple child parallel groups(i.e. $G_{parent} = \bigcup_i G_{child_i}$), the child parallel group with the smallest number of nodes is removed from parallel group set. This pruning step removes less impactful parallel groups in parent-child structures, while preserving sufficient optimization space within each parent-level parallel path.

3.2.2 Search with Predicted Time Distribution. We utilizes a multi-round iterative process as illustrated in Figure 5: in each round, distinct cost limits are allocated to every parallel group, and an optimal configuration is systematically explored within the bounds of these cost limits. Specifically, the following strategies are employed:

Allocate cost recursively: ConfigNavigator employs a bottom-up recursive cost allocation strategy, assigning costs to parallel groups from child to parent groups. When allocating resources to a parent group, the algorithm specifically excludes computational nodes that have already been assigned to its child groups For example, consider a parent group $G_m = \{(v_1, v_2, v_3), (v_4, v_5)\}$ with cost bound C_{bound} and a child group $G_n = \{(v_2), (v_3)\}$. After assigning a cost $C_n \in (0, C_{\text{bound}})$ to G_n, the remaining nodes in G_m, i.e., $\{v_1, v_4, v_5\}$, are allocated a cost $C_{m\setminus n} \in (0, C_{\text{bound}} - C_n)$. This approach ensures that costs are distributed efficiently without redundancy across nested groups. For each iteration, we generate different costs allocations for parallel groups.

Using previous configurations as initial values: After each search, ConfigNavigator stores the resource configuration, cost, and time distribution of the parallel group. During subsequent searches, the configuration with the closest cost to the current group is retrieved and used as the initial value.

Hybrid Search: ConfigNavigator combines Bayesian optimization and exhaustive search for exploring the search space within parallel groups. While exhaustive search is suitable for groups with limited functions, the complexity becomes prohibitive for larger groups. For such cases, Bayesian optimization is employed to focus on promising regions of the search space, reducing unnecessary search costs and accelerating convergence. This hybrid approach balances thoroughness and efficiency, leveraging predictions from a simple neural network to further narrow the search space.

Equivalence Transformation: ConfigNavigator estimates the time and cost of all functions within a parallel group with Monte-Carlo method, after that it replaces the parallel group with an equivalent function. This equivalent function combines the following properties: A fixed execution cost, which is equal to the total cost allocated to the parallel group. Time distribution under this total cost, generated through Monte Carlo simulation to capture the aggregated behavior of the parallel group. By replacing parallel groups with equivalent functions, ConfigNavigator significantly reduces the complexity of subsequent calculations.

3.3 Online Configuration Adjustment

3.3.1 Critical-Path Aware Configuration Tuning. ConfigNavigator proposes an integration of the online adjustment mechanism capable of dynamically adapting pipeline configurations due to SLO requirements. It conducts a localized search within the neighborhood field of the optimal configurations identified offline. The advantage of this approach is that it avoids falling into poor configurations

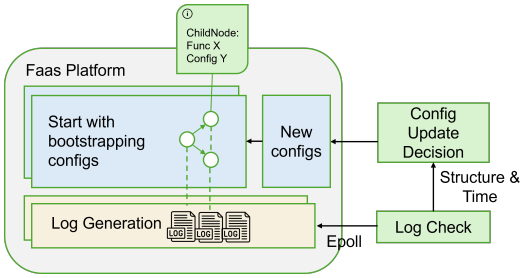


Figure 6: Online Implementation.

during the search, which could result in significant increases in execution time and costs.

When costs exceed the budget: Non-critical path constriction. Adjusting the resource configurations of non-critical paths does not increase the end-to-end latency of the pipeline but can significantly reduce execution costs. ConfigNavigator employs a *batch adjustment* strategy for functions that are not located on the same non-critical path. By simultaneously reducing the resources allocated to multiple functions on non-critical paths, this approach minimizes the need for frequent, individual adjustments while maintaining overall system efficiency.

The adjustment granularity is given by:

$$\Delta r = \alpha(C_{real} - C_{bound}),$$

where α is the adjustment factor that determines the amplitude for the batch of function configurations to be adjusted.

When costs are below the budget: Critical path relaxation. When costs are below the budget, ConfigNavigator focuses on critical path relaxation to enhance pipeline performance. Adjusting resource configurations on the critical path directly impacts end-to-end latency. However, large-scale adjustments can introduce instability by altering the critical path itself. To mitigate this, ConfigNavigator adopts a conservative approach, making small, incremental changes to individual functions. This ensures that performance gains are achieved without disrupting pipeline stability.

3.3.2 When pipeline structure is updated. Upon pipeline updates, ConfigNavigator performs basic profiling for potential new modules and re-applies search strategy on the updated pipeline. Both phases exhibit exceptional computational efficiency, with complete execution times consistently remaining below 100 milliseconds. This rapid adaptation capability enables ConfigNavigator to dynamically respond to structural pipeline modifications.

3.3.3 Online Implementation. ConfigNavigator is implemented on AWS Lambda [5], a widely-used Function-as-a-Service (FaaS) platform [3]. To ensure platform independence, ConfigNavigator avoids relying on platform-specific APIs for modifying function configurations, invoking functions, and retrieving execution logs. These APIs, such as AWS Lambda's *updateFunctionConfiguration* and *getFunctionLogs*, are fundamental features of most serverless platforms. Thus, ConfigNavigator can be easily extended to other serverless platforms with minimal effort, making it highly adaptable for diverse deployment environments.

Figure 6 illustrates the implementation design of ConfigNavigator's online component, which consists of two non-intrusive modules: the log inspection module and the configuration update module. For pipelines running on commercial serverless platforms, the specific execution process is typically a black box. To address this challenge, the log inspection module periodically queries execution logs to extract information about end-to-end latency and individual function execution times. By analyzing latency data and call details from logs, the log inspection module reconstructs the pipeline structure (e.g., certain modules may or may not be invoked, or may be called a varying number of times) and determines whether the pipeline exceeds its cost constraints. The configuration update module identifies functions requiring configuration updates and applies changes without disrupting ongoing pipeline executions. Rather than directly modifying configurations of running functions, updates are made to configuration files stored in centralized storage (e.g., AWS S3). Each serverless function reads from this centralized storage to obtain configurations for its subfunctions, then invokes the corresponding subfunctions accordingly.

4 Evaluation

4.1 Experiment Setup

Dataset from Video Sharing Platforms: We collect 50 most popular videos from the Music section of Bilibili in Nov. 13th, 2024. The dataset comprises recordings captured using different camera types and under distinct lighting conditions, with resolutions ranging from 360p to 1080p. To facilitate subsequent analysis and processing, the videos are segmented into 5-second clips. All clips are stored on Amazon S3 [4], leveraging its high scalability and reliability to ensure efficient data management. The input streaming patterns are simulated using production traces from Microsoft Azure [38] to reflect realistic workload scenarios.

Serverless Video Processing Pipelines: We evaluate our solution on three different pipelines. The first pipeline is face detection pipeline similar to the one tested in Charmseeker [52] (license plate recognition module replaced with HD module). We also select two representative video processing pipelines from Bilibili [8] and Google MediaPipe [1]: face beautification [9] and Autoclip [16]. Both pipelines are re-implemented in a serverless version and deployed on AWS Lambda. The specific functions in these three pipelines are listed in Table 3. Notably, the structure of face-beautification pipeline would change according to number of faces detected in the pipeline.

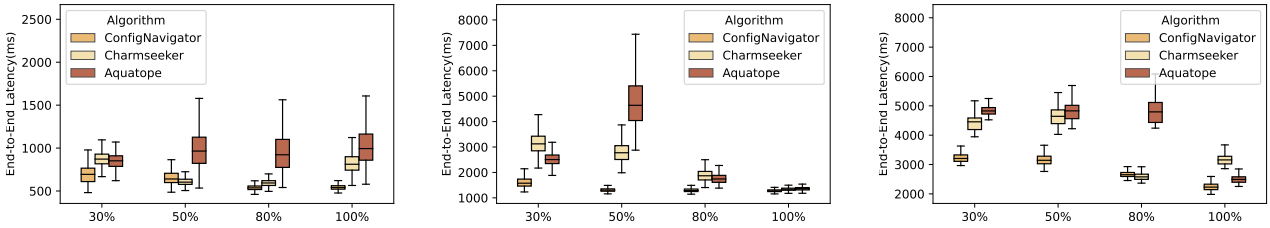
Alternative Solutions: We compare our solution with specialized versions of state-of-the-art methods: (1) Charmseeker [52] solution, which directly addresses the problem of configuration optimization under budget constraints, using a two-stage Bayesian search to find the best pipeline configuration online. (2) Aquatope [53], which uses a special noise modeling approach and batch Bayesian search. We perform 40 search iterations for each of the two baselines, which is twice the number used in Charmseeker.

4.2 Overall Performance

For different application pipelines and cost constraints, ConfigNavigator exhibits robust scalability across this complexity spectrum as illustrated in Figure 7, maintaining its performance advantages in both simple and complex scenarios. We observe improvements

Table 3: Video pipelines used for evaluating ConfigNavigator and their tasks.

Pipeline	Description	Number	Tasks	Parallel Depth
face-detection	detect human face	4	decode, face detection, high resolution, encode	1
face-beautification	perform face beautification on face extracted	7	decode, ROI, face beautification, judge resolution, HD, restore, encode	2
Autoclip	clip video to ideal size based on objects and borders	8	decode, scale image, border detection, face detection, object detection, signal fusing, scene cropping, encode	3



(a) face-detection (b) face-beautification (c) Autoclip

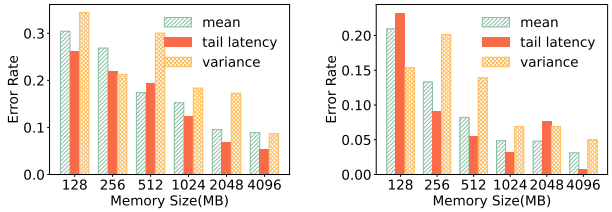
Figure 7: Overall Performance. The x-axis denotes the threshold for cost bounds exceeding the minimal cost percentile.

Table 4: ConfigNavigator improves both time and cost consumption in pipeline Autoclip.

Cost Bound	30%	50%	80%	100%
Latency	**28.34**	**31.23**	-3.95	0.25
Cost	0.53	1.77	**10.35**	-1.73

in three aspects: (1) ConfigNavigator performs better in resource-constrained scenarios. Under the same cost constraint, ConfigNavigator achieves 20-50% lower end-to-end application latency in most cases compared to the best baseline method between Charmseeker and Aquatope. (2) ConfigNavigator not only improves average performance but also reduces performance variance: the whiskers in ConfigNavigator's boxplots are noticeably shorter than those of the baselines. (3) Notably, the performance gains are most pronounced at moderate workload levels (50%), where our system's intelligent configuration optimization shows its greatest impact. As workloads approach 100%, the performance differential between different approaches diminishes, suggesting that system resource saturation becomes the dominant factor rather than configuration efficiency.

To summarize the improvements of our method in end-to-end latency and cost, Table 4 presents ConfigNavigator's performance gains compared to the better-performing baseline. The table shows that ConfigNavigator achieves significant latency reductions under stricter cost constraints. When the cost constraint is 30% or 50% above the minimal possible cost, ConfigNavigator reduces latency by approximately 30% without exceeding the cost incurred by the baselines. Under higher cost constraints, the improvements by ConfigNavigator are less pronounced. This is primarily because higher cost constraints reduce the difficulty of the search for the baseline methods; less precise search results do not lead to significant changes in the final outcomes (i.e., with higher cost constraints, there exists a broad feasible region where the shortest achievable latency is close to the optimal solution).



(a) Real function set. (b) Validation set.

Figure 8: Error of prediction model evaluated on both real function set and validation set.

Table 5: Prediction error(%) for functions in face beautification pipeline.

Resource(MB)	128	256	512	1024	2048	4096
face detection	37.5	29.7	20.4	17.6	16.9	12.3
face beautification	8.9	7.3	5.6	5.8	5.1	3.9
restore	27.8	23.4	15.6	10.9	12.7	12.3
HD	9.7	9.5	6.7	4.3	4.7	3.9

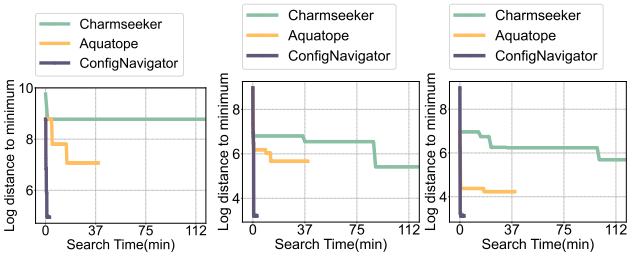
4.3 The Performance of Prediction Model

To evaluate the performance of the prediction model under different functions and memory configurations, we conduct a series of micro-benchmarks. Our function test set includes two parts: all the functions in the pipeline mentioned above and the test set generated from the dataset in § 3.1 (split in an 8:2 ratio). As shown in Figure 8, when we increase the configured memory, the error rates for average latency, 95% latency, and variance all decreased.

Table 5 shows the predication error of some important functions used in the application pipeline. As seen, for functions that do not involve machine learning, the prediction error rates are relatively small, with average latency prediction errors being less than 10%. This evaluation demonstrates that our prediction model achieves practical utility across diverse video processing scenarios, with particularly strong performance in environments with adequate

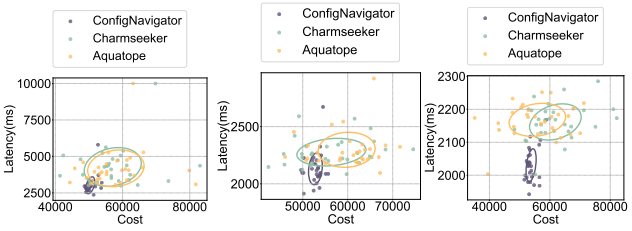
memory resources. The analysis provides a solid foundation for both deployment decisions and future research directions in serverless video processing optimization.

4.4 Effectiveness of the Offline Search



(a) budget 30% above minimal cost. **(b) budget 50% above minimal cost.** **(c) budget 100% above minimal cost.**

Figure 9: The convergence process of ConfigNavigator compared to baseline methods.



(a) budget 30% above minimal cost. **(b) budget 50% above minimal cost.** **(c) budget 100% above minimal cost.**

Figure 10: Search effect under various budget constraints (30%, 50%, and 100% Above Minimal Cost). Cost is measured in MB · ms, and the pricing may vary across different platforms.

We evaluate the effectiveness of our offline search algorithm with fixed input. We repeat putting the same 5-second clips into evaluation pipelines, ensuring the structure and execution of each function are stable. We have both comparison algorithms, Charmseeker and Aquatope, perform online searches 80 times each (The original number of searches in Charmseeker's evaluation is 20, but we quadruple it due to our more complex pipeline structure. For each search, video streams are continuously sent to the pipeline entrance for approximately 30 seconds), while ConfigNavigator directly searches locally after offline testing of metrics for each function in the pipeline.

The search convergence curves for the three functions are shown in Figure 9. The search curve demonstrates that ConfigNavigator reduces 94.11% of time compared to the batch Bayesian configuration optimizer Aquatope, while achieving superior performance. This highlights ConfigNavigator's effective integration of a prediction model with an offline search algorithm.

As observed in Figure 10, ConfigNavigator consistently identifies configurations that outperform both baselines (Charmseeker and Aquatope) in terms of cost and latency. Although ConfigNavigator does not always discover the optimal configuration due to discrepancies between the prediction model and real-world environment, it demonstrates more stable search behavior, indicating

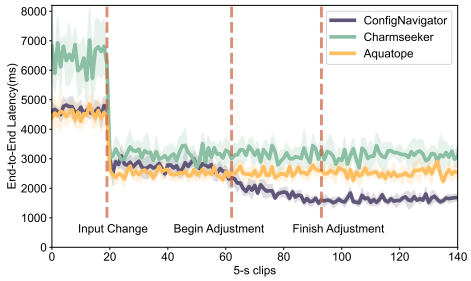


Figure 11: End-to-end latency of pipeline face beautification change when input video changes.

efficient exploration of the search space. As observed, although ConfigNavigator does not always find the best configuration due to the difference between predict model and real-world environment, its search capability is more stable, indicating efficient exploration of the solution space. In contrast, the two comparison algorithms sometimes find better results but lack consistency in their search performance. We observe from Figure 10 that ConfigNavigator consistently outperforms the baseline methods under constrained budget scenarios. Specifically, when the cost bound is set at 30% or 50% above the minimal cost, ConfigNavigator achieves an average improvement of 31.57% in pipeline latency and 14.31% in cost efficiency compared to Charmseeker and Aquatope. When the cost bound is set at 100% above the minimum (representing a less constrained scenario), all three methods converge to comparable performance levels(also demonstrated in Figure 7). This suggests that ConfigNavigator's advantage is particularly pronounced when operating under tighter resource constraints.

4.5 Tuning in dynamic environment

Figure 11 demonstrates the system's latency response to input resolution changes. Upon detecting lower resolution input, ConfigNavigator initiates adaptation. When processing dynamic video inputs, ConfigNavigator reduces end-to-end latency by 13.97% across applications in average. The system successfully establishes a lower steady-state latency with reduced variance after adaptation.

5 Conclusion

Video processing pipelines present significant challenges in resource management due to time-consuming measurement, large configuration spaces, and irregular input stream and pipeline structure. To address these challenges, we introduce ConfigNavigator, an innovative resource tunning framework adapting to both fluctuating input streams and dynamic pipeline structures. ConfigNavigator uses a prediction model based on basic operation profiling to reduce measurement costs, combined with an efficient search algorithm and dynamic adjustment capabilities. Through experiments on real-world dataset, we demonstrate the superiority of ConfigNavigator.

Acknowledgments

We sincerely thank our anonymous reviewers and labmates in the Routing Group from Tsinghua University for their valuable feedback. This work is sponsored by the National Natural Science Foundation of China (No. 62221003) and Hong Kong RGC (No. 26212525). Zili Meng and Mingwei Xu are corresponding authors.

References

[1] Google AI. 2023. MediaPipe. https://github.com/google-ai-edge/mediapipe. Accessed: 2025-1-10.

[2] Nabeel Akhtar, Ali Raza, Vatche Ishakian, and Ibrahim Matta. 2020. COSE: Configuring Serverless Functions using Statistical Learning. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 1–9.

[3] Amazon Web Services. 2023. Amazon API Gateway Developer Guide. https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html. Accessed: 2024-10-10.

[4] Amazon Web Services. 2023. Amazon S3 - Cloud Object Storage. https://aws.amazon.com/s3/. Accessed: 2024-10-10.

[5] Amazon Web Services. 2025. AWS Lambda Developer Guide. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html. Accessed: 2025-1-10.

[6] Lixiang Ao, Liz Izhikevich, Geoffrey M Voelker, and George Porter. 2018. Sprocket: A serverless video processing framework. In *Proceedings of the ACM Symposium on Cloud Computing*. 263–274.

[7] Christos G. Bampis, Zhi Li, Ioannis Katsavounidis, Te-Yuan Huang, Chaitanya Ekanadham, and Alan C. Bovik. 2021. Towards Perceptually Optimized Adaptive Video Streaming—A Realistic Quality of Experience Database. *IEEE Transactions on Image Processing* 30 (2021), 4449–4464.

[8] Bilibili. 2025. Bilibili. https://www.bilibili.com/. Accessed: 2025-1-10.

[9] Bilibili Technology. 2023. Bilibili Technology Article. https://www.bilibili.com/opus/861103172928667703. Accessed: 2025-1-10.

[10] Netflix Tech Blog. 2019. Unbundling Data Science Workflows with Metaflow and AWS Step Functions. https://netflixtechblog.com/unbundling-data-science-workflows-with-metaflow-and-aws-step-functions-d454780c6280. Accessed: 2025-01-06.

[11] Netflix Tech Blog. 2020. Rebuilding Netflix Video Processing Pipeline with Microservices. https://netflixtechblog.com/rebuilding-netflix-video-processing-pipeline-with-microservices-4e5e6310e359. Accessed: 2025-01-06.

[12] Shen Cai et al. 2023. Cost-Efficient Serverless Inference Serving with Joint Batching and Multi-Processing. In *Proceedings of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems*. ACM.

[13] Simon Eismann, Long Bui, Johannes Grohmann, Cristina Abad, Nikolas Herbst, and Samuel Kounev. 2021. Sizeless: Predicting the optimal size of serverless functions. In *Proceedings of the 22nd International Middleware Conference*. 248–259.

[14] Sadjad Fouladi, Riad S Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, fast and slow:{Low-Latency} video processing using thousands of tiny threads. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 363–376.

[15] Apostolos Galanopoulos, Jose A Ayala-Romero, Douglas J Leith, and George Iosifidis. 2021. AutoML for video analytics with edge computing. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 1–10.

[16] Google AI. 2023. MediaPipe AutoFlip Documentation. https://github.com/google-ai-edge/mediapipe/blob/master/docs/solutions/autoflip.md. Accessed: 2024-10-10.

[17] Emily M. Hand, Carlos D. Castillo, and Rama Chellappa. 2018. Predicting facial attributes in video using temporal coherence and motion-attention. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE.

[18] Jiawei He et al. 2018. Probabilistic video generation using holistic attribute control. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[19] M. Reza HoseinyFarahabady, Albert Y. Zomaya, and Zahir Tari. 2017. A model predictive controller for managing QoS enforcements and microarchitecture-level interferences in a lambda platform. *IEEE Transactions on Parallel and Distributed Systems* 29, 7 (2017), 1442–1455.

[20] Biao Hou, Song Yang, Fernando A Kuipers, Lei Jiao, and Xiaoming Fu. 2023. Eavs: Edge-assisted adaptive video streaming with fine-grained serverless pipelines. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 1–10.

[21] Qi Huang. 2020. Evolution of the infrastructure of video era. https://segmentfault.com/a/1190000041045162. Accessed: 2025-01-06.

[22] Lianchen Jia, Chao Zhou, Tianchi Huang, Chaoyang Li, and Lifeng Sun. 2023. Rdladder: Resolution-duration ladder for vbr-encoded videos via imitation learning. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 1–10.

[23] Lianchen Jia, Chao Zhou, Tianchi Huang, Chaoyang Li, and Lifeng Sun. 2024. Dancing with Shackles, Meet the Challenge of Industrial Adaptive Streaming via Offline Reinforcement Learning. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2169–2178.

[24] Chao Jin, Zili Zhang, Xingyu Xiang, Songyun Zou, Gang Huang, Xuanzhe Liu, and Xin Jin. 2023. Ditto: Efficient Serverless Analytics with Elastic Parallelism. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 406–419. https://doi.org/10.1145/3603269.3604816

[25] Zhengyu Lei, Xiao Shi, Cunchi Lv, Xiaobing Yu, and Xiaofang Zhao. 2023. Chitu: accelerating serverless workflows with asynchronous state replication pipelines. In *Proceedings of the 2023 ACM symposium on cloud computing*. 597–610.

[26] Yongkang Li, Yanying Lin, Yang Wang, Kejiang Ye, and Chengzhong Xu. 2022. Serverless computing: state-of-the-art, challenges and opportunities. *IEEE Transactions on Services Computing* 16, 2 (2022), 1522–1539.

[27] Zijun Li, Quan Chen, Shuai Xue, Tao Ma, Yong Yang, Zhuo Song, and Minyi Guo. 2020. Amoeba: QoS-Awareness and Reduced Resource Usage of Microservices with Serverless Computing. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 399–408.

[28] Zhen Li, Yuchen Li, Yuxiang Li, Yingshu Li, and Min Li. 2024. Joint Optimization of Parallelism and Resource Configuration for Serverless Function Steps. *IEEE Transactions on Parallel and Distributed Systems* 35, 1 (2024), 123–137.

[29] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Sameh Elnikety, Somali Chaterji, and Saurabh Bagchi. 2022. ORION and the Three Rights: Sizing, Bundling, and Prewarming for Serverless DAGs. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, 303–320.

[30] Anupama Mampage, Shanika Karunasekera, and Rajkumar Buyya. 2021. Deadline-aware Dynamic Resource Management in Serverless Computing Environments. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 483–492. https://doi.org/10.1109/CCGrid51090.2021.00058

[31] Haoran Qiu, Weichao Mao, Archit Patke, Chen Wang, Hubertus Franke, Zbigniew T. Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. 2022. SIMPPO: A Scalable and Incremental Online Learning Framework for Serverless Resource Management. In *Proceedings of the 13th ACM Symposium on Cloud Computing (SoCC)*. 1–14. https://doi.org/10.1145/3542929.3563475

[32] Haoran Qiu, Weichao Mao, Chen Wang, Hubertus Franke, Alaa Youssef, Zbigniew T. Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. 2023. AWARE: Automate Workload Autoscaling with Reinforcement Learning in Production Cloud Systems. In *Proceedings of the 2023 USENIX Annual Technical Conference (USENIX ATC 2023)*. USENIX Association, 387–402. https://www.usenix.org/conference/atc23/presentation/qiu-haoran

[33] Ali Raza, Nabeel Akhtar, Vatche Isahagian, Ibrahim Matta, and Lei Huang. 2023. Configuration and Placement of Serverless Applications Using Statistical Learning. *IEEE Transactions on Network and Service Management* 20, 2 (2023), 1065–1077.

[34] Francisco Romero, Mark Zhao, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. Llama: A heterogeneous & serverless framework for auto-tuning video analytics pipelines. In *Proceedings of the ACM symposium on cloud computing*. 1–17.

[35] Gor Safaryan, Anshul Jindal, Mohak Chadha, and Michael Gerndt. 2022. SLAM: SLO-Aware Memory Optimization for Serverless Applications. In *Proceedings of the 15th IEEE International Conference on Cloud Computing (CLOUD)*. 1–10. https://doi.org/10.1109/CLOUD52976.2022.00011

[36] sandvine. 2024. global-internet-phenomena-report-2024. https://www.sandvine.com/global-internet-phenomena-report-2024. [Online; accessed 30-May-2024].

[37] Amazon Web Services. 2025. Live Streaming on AWS. https://aws.amazon.com/cn/solutions/implementations/live-streaming-on-aws/. Accessed: 2025-01-06.

[38] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX annual technical conference (USENIX ATC 20)*. 205–218.

[39] Junxian Shen, Han Zhang, Yantao Geng, Jiawei Li, Jilong Wang, and Mingwei Xu. 2022. Gringotts: Fast and Accurate Internal Denial-of-Wallet Detection for Serverless Computing. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 2627–2641. https://doi.org/10.1145/3548606.3560629

[40] Mikhail Shilkov. 2024. Cold Starts in Serverless Functions. https://mikhail.io/serverless/coldstarts/ Accessed: 2024-12-17.

[41] Paulo Silva, Daniel Fireman, and Thiago Emmanuel Pereira. 2020. Prebaking Functions to Warm the Serverless Cold Start. In *Proceedings of the 21st International Middleware Conference*. ACM, 1–13.

[42] O. Sinnen and L. A. Sousa. 2005. Communication Contention in Task Scheduling. *IEEE Transactions on Parallel and Distributed Systems* 16, 6 (2005), 503–515. https://doi.org/10.1109/TPDS.2005.64

[43] TikTok. 2025. Effect House. https://effecthouse.tiktok.com/ Accessed: 2025-04-11.

[44] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking behind the curtains of serverless platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, 133–146.

[45] Yilun Wang, Pengfei Chen, Hui Dou, Yiwen Zhang, Guangba Yu, Zilong He, and Haiyu Huang. 2024. FaaSConf: QoS-aware Hybrid Resources Configuration for Serverless Workflows. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1–12. https://doi.org/10.

1109/ASE52444.2024.00012

[46] Jinfeng Wen, Zhenpeng Chen, Xin Jin, and Xuanzhe Liu. 2023. Rise of the Planet of Serverless Computing: A Systematic Review. *ACM Trans. Softw. Eng. Methodol.* 32, 5, Article 131 (July 2023), 61 pages. https://doi.org/10.1145/3579643

[47] Zhaojie Wen, Yishuo Wang, and Fangming Liu. 2022. Stepconf: Slo-aware dynamic resource configuration for serverless function workflows. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 1868–1877.

[48] Haoning Wu, Erli Zhang, Liang Liao, Chaofeng Chen, Jingwen Hou, Annan Wang, Wenxiu Sun, Qiong Yan, and Weisi Lin. 2023. Exploring video quality assessment on user generated contents from aesthetic and technical perspectives. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 20144–20154.

[49] Zhe Yang, Phuong Nguyen, Haiming Jin, and Klara Nahrstedt. 2019. MIRAS: Model-based reinforcement learning for microservice resource allocation over scientific workflows. In *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*. IEEE, 122–132.

[50] Tianming Zang, Ce Zheng, Shiyao Ma, Chen Sun, and Wei Chen. 2023. A general solution for straggler effect and unreliable communication in federated learning. In *ICC 2023-IEEE International Conference on Communications*. IEEE, 1194–1199.

[51] Fang-Lue Zhang, Xian Wu, Rui-Long Li, Jue Wang, Zhao-Heng Zheng, and Shi-Min Hu. 2018. Detecting and removing visual distractors for video aesthetic enhancement. *IEEE Transactions on Multimedia* 20, 8 (2018), 1987–1999.

[52] Miao Zhang, Yifei Zhu, Jiangchuan Liu, Feng Wang, and Fangxin Wang. 2022. Charmseeker: Automated pipeline configuration for serverless video processing. *IEEE/ACM Transactions on Networking* 30, 6 (2022), 2730–2743.

[53] Zhuangzhuang Zhou, Yanqi Zhang, and Christina Delimitrou. 2022. Aquatope: Qos-and-uncertainty-aware resource management for multi-stage serverless workflows. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 1–14.