National College of Ireland

Optimising Serverless Data Ingestion: Evaluating the Impact of AWS Lambda Configuration on Performance and Cost in Amazon S3 Data Lakes

MSc Research Project

Cloud Computing

Abhishek Selvadurai

Student ID: 23416882

School of Computing

National College of Ireland

Supervisor: Ahmed Makki

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name:	Abhishek Selvadurai		
Student ID:	23416882		
Programme:	Cloud Computing	**Year:**	2025-2026
Module:	MSc Research Project		
Supervisor:	Ahmed Makki		
Submission Due Date:	11-12-2025		
Project Title:	Optimising Serverless Data Ingestion: Evaluating the Impact of AWS Lambda Configuration on Performance and Cost in Amazon S3 Data Lakes		
Word Count:	7808	**Page Count:**	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Abhishek Selvadurai

Date: 11-12-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on the computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Optimising Serverless Data Ingestion: Evaluating the Impact of AWS Lambda Configuration on Performance and Cost in Amazon S3 Data Lakes

Abhishek Selvadurai
23416882

Abstract

Optimising serverless data ingestion is increasingly critical for modern, cost-aware cloud data platforms that depend on AWS Lambda and Amazon S3 for scalable, event-driven intake. This study conducts a large-scale empirical evaluation of three key Lambda configuration parameters—memory allocation, batch/part size, and reserved concurrency—and assesses their combined effects on performance and cost. Using a 3×3×3 full factorial design, the work evaluates 54 configurations and 27,000 invocations across both event-driven and multipart ingestion modes. Results show that memory allocation is the leading determinant of performance, with up to 70% p95 latency reduction (from 512 MB to 2048 MB). Across configurations, cold-start duration was insignificant, with values ranging between 0.33 and 0.47 s, which implies that the cost of a cold start is dominated by runtime initialization rather than characteristics of the workload. Batch size and part size imposed moderate overhead due to S3 requests being staged in this way; reserved concurrency had only a weak effect, marginally smoothing throughput. The higher memory tiers had higher costs per millisecond but were the most efficient with regard to overall cost due to reduced execution time. Here, a valid configuration decision matrix for tuning ingestion-heavy Lambda pipelines is given, and some interesting future directions for research might include multi-region evaluation, runtime comparisons, and real-world testing of ingestion patterns, along with adaptive optimization strategies.

1. Introduction

In modern data platforms, serverless computing is at the core when considering workloads with elastic, event-driven needs for ingesting millions of objects into cloud storage. AWS Lambda is arguably the most commonly used execution environment, given that its configuration directly impacts latency and throughput characteristics, not to mention total operating costs.

1.1 Background

AWS Lambda helps users build applications that run on demand when events occur, as well as applications that run from a queue of events. AWS Lambda supports the construction of scalable, on-demand Serverless Ingestion Pipelines that are charged on a consumption basis across cloud storage and computing resources. Ingestion workloads are generally made up of thousands of short-lived function invocations that process object uploads and often involve multipart storage transactions for large files. Although AWS Lambda makes building Serverless Ingestion

Pipelines much easier, it does present issues with unreliable performance and unpredictable latencies/cost due to the way that it operates and how you configure AWS Lambda's runtime parameters (memory allocation, batch size, concurrency).

1.2 Problem Statement

Previous research does not provide any multi-dimensional evidence surrounding the use of Lambda. In addition, none provide insight into the significance of p95 latency, queueing behaviour or cost to performance interactions when the configurations change together which reduces confidence in optimising the ingestion pipelines.

1.3 Motivation

Every single day, the volume of input to cloud services exceeds a million items, and the manner in which teams optimise the configuration parameters for AWS Lambda is largely determined by heuristics. The AWS Lambda pricing model takes into account both execution time and memory allocated to the function, so these configurations have a direct influence on cost and latency. Consequently, we need to accurately define these trade-offs, as part of the definition of "empirical study" requires characterisation of any such trade-offs. Therefore, an empirical research methodology must be developed that will allow for controlled testing of AWS Lambda function parameter configuration with specific ingestion workloads.

1.4 Research Objectives

This work defines the problem at hand and outlines its objectives as follows:

1. assess the trade-off between memory allocation and P95 latency, cold-start duration, throughput stability, and cost efficiency
2. Evaluate the performance impact of batch size with regards to Amazon S3 part size for both event-driven and batch paradigms of ingestion.
3. Analyze how reserved concurrency affects throughput stability and queueing behavior.
4. Identify significant main effects by using one-way ANOVA and Tukey HSD post hoc analyses.
5. Develop a configuration decision matrix that will help tune Lambda ingestion pipelines.

1.5 Research Question

How do variations in memory allocation, batching/part size, and reserved concurrency affect latency, cold-start overhead, throughput stability, and compute cost in AWS Lambda–based ingestion pipelines for Amazon S3?

1.6 Assumptions

It was assumed that AWS Lambda had low variability in performance within the selected region, eu-west-1, and any variation due to co-tenant activity would not significantly impact the averages reported. It was also assumed that all memory tiers used with the Python runtime would perform uniformly within the Lambda execution environment. Finally, the event-driven and batch workloads were considered to be representative of real-world ingestion pipelines

processing medium- to large-sized files. External services were assumed to have a negligible performance impact, to include Amazon S3 and CloudWatch operating within their normal latency profiles for the configuration used in this study.

1.7 Limitations

Results are only generalizable to other regions, runtimes, or service categories because only one AWS region (eu-west-1) and one Python 3.x runtime were examined. The ingestion sources like Kinesis or DynamoDB Streams are not considered. Synthetic workloads were utilized instead of live data in production, hence reducing ecological validity. The research studied three configuration parameters (only); however, there are multiple configuration parameters to consider that play an important role in evaluating Virtual Private Cloud network support via Cloud Security-Performance Policies, and Non-native Runtime Types.

1.8 Contributions

This research is the first empirical attempt to quantify Lambda's ingestion performance with control for several variables, in the form of 54 configurations and more than 27,000 invocation calls. We will provide realistic ingestion performance patterns and their cost implications by evaluating the data with ANOVA and Tukey HSD analysis methods to disclose the effects of the various configurations. Additionally, our research will create a decision matrix that practicalizes if not trivializes how to go about choosing the best configuration for ingestion performance.

1.9 Structure of the Report

Section 2 reviews the key literature providing the foundation of this research. Section 3 outlines the framework of methodologies utilized in this study. Section 4 presents the architecture, and Section 5 describes the implementation details. Section 6 reports on the results, while Section 7 discusses and contextualizes them. Section 8 concludes the findings with a summary of the major contributions that this study makes.

2. Literature Review

This paper contributes to the relatively fragmented body of work on AWS Lambda performance in the context of the broader move to serverless computing. Interest in this space has risen, but previous work tends to examine configuration parameters one at a time, which limits its applicability to real workloads where compute, batching, and concurrency influence each other. The next section reviews existing literature and highlights the specific gaps this study aims to fill.

2.1 Memory and Configuration Controls

Baldini et al. 2017; Jonas et al. 2019 correctly demonstrate that characteristics of memory impact both computational power and latency. However, these analyses are one-dimensional and ignore any combined effects related to the structure of the workload and concurrency. Hence, although their theoretical contributions are instructive, they do not lead to actionable guidelines for tuning ingestion pipelines and thus demonstrate a need for controlled confirmation. This directly motivates the explicit inclusion of memory as a main effect in our factorial design.

2.2 Cold Start Behaviour

This study firmly establishes the relationship between runtime characteristics and initialization overhead, thereby extending findings from previous cold-start studies by Wang et al. (2018). However, the results did not support previous work in those settings where ingestion-heavy workloads generate bursty activity that may be used to make cold-start phenomena more pronounced. The current measurements show the exact manifestations of cold start in all the measured setups and contradict the previous claims of generality by highlighting the boundaries within which their validity holds. This directly motivated measuring cold-starts under ingestion-heavy workloads in this study.

2.3 Concurrency and Scheduling

Shahrad et al. (2020, USENIX ATC) provide valuable insights at the infrastructure level on concurrency scaling. However, they do not assess the implications for application-level performance in terms of latency or for the stability of data ingestion.The work presented herein shows that reserved concurrency has only a negligible effect, with only minor stabilization benefits that were not captured by previous work.

2.4 Batching and Multipart Upload Performance

Although multipart upload studies (Gadban & Kunkel, 2021) explain trade-offs in storage throughput, they do not address the dynamics involved in Lambda execution. Hence, their conclusions are incomplete for an ingestion system that couples compute and store. This study confirms that part size influences S3 overhead more than Lambda compute, thus filling that omission. This gap informed the choice of 8–64 MB part sizes in the experiments.

2.5 Recent Optimisation Research

While industry studies recognise the value of Lambda optimisation, most remain heuristic and ingestion-agnostic. Recent empirical work has begun to explore tuning models, but validation is still limited in scope. Karamzadeh and Shameli-Sendi (2024) show that lightweight virtualisation can reduce cold-start delay, yet the gains are bounded and evaluated outside ingestion-heavy workloads, which aligns with this study's finding that relative cold-start behaviour remains stable across configurations. Bluemke and Zdanowski (2025) empirically evaluate multiple Lambda configurations and confirm that memory size strongly shapes cost–performance trade-offs, while Ustiugov et al. (2021) provide empirical cold-start benchmarking and show bounded improvements through mitigation techniques. Bodner et al. (2025) benchmark serverless infrastructure at scale and identify clear performance and cost boundaries for data processing workloads. Together, these works demonstrate growing interest in empirical optimisation but still lack multi-factor, ingestion-focused evaluation — the gap this research addresses.

2.6 Identified Gaps (Synthesised)

Identified Gap	Why It Matters	What Existing Studies Miss

Single-factor evaluation	Real workloads involve interacting parameters	Cannot explain combined effects of memory, batching, and concurrency
Compute and storage studied separately	Ingestion pipelines couple Lambda execution with S3 behaviour	Storage-only or compute-only results fail to predict real ingestion performance
Lack of statistical validation	Heuristic tuning produces unreliable optimisation	Findings are not tested through controlled or inferential methods
Cold-start and concurrency assumptions not contextualised	Behaviour may differ under ingestion workloads	Prior work generalises without testing in ingestion scenarios
No practical configuration guidance	Practitioners lack actionable tuning support	Studies analyse behaviour but do not translate findings into decision frameworks

Table 1. Gaps Identified

These gaps represent the need for a controlled, multi-factor study to investigate realistic patterns of ingestion and to statistically validate configuration effects.

2.7 How This Study Addresses the Gaps

It addresses the identified gaps by jointly assessing memory, batch/part size, and reserved concurrency through a 3×3×3 factorial design applied to event-driven and multipart S3 ingestion workloads. Main effects are derived via ANOVA and Tukey HSD, producing an evidence-based configuration decision matrix. Since prior work relied on heuristic rather than statistically validated results, this study uses these analyses to deliver rigorously supported insights. The findings confirm memory as the principal optimisation lever, reinterpret its impact as configuration-dependent, and show a small but stabilising influence of concurrency while verifying consistent cold-start behaviour under ingestion conditions. It therefore represents the first statistically grounded, multi-factor analysis of Lambda-based ingestion performance and cost. This synthesis demonstrates why a multi-factor empirical evaluation is essential to produce reliable configuration guidance for real-world ingestion-heavy workloads.

Study	**Strength**	**Critical Limitation**	**Relevance to This Study**
Bluemke & Zdanowski, 2025	Empirical evaluation of Lambda configuration choices	Not ingestion-specific	Supports inclusion of configuration tuning dimensions

Bodner et al., 2025	Large-scale empirical benchmarking of serverless performance and cost	Focuses on data processing, not ingestion workloads	Reinforces cost-performance findings
Karamzadeh & Shameli-Sendi, 2024	Demonstrates empirical cold-start reduction technique	Evaluated outside ingestion workloads	Supports conclusions on bounded cold-start variation
Marcelino & Nastic, 2024	Investigates data-passing optimisation in serverless settings	Focuses on transfer patterns, not ingestion pipelines	Reinforces need for broader multi-factor evaluation
Ustiugov et al. (2021)	Real cold-start performance benchmarking	Evaluates cold-start only, not throughput or ingestion context	Supports this study's finding that cold-start behaviour remains stable across configurations
(**Gadban & Kunkel, 2021**)	Good S3 multipart throughput analysis	Ignores Lambda interaction	Motivates Lambda–S3 evaluation
Shahrad et al., 2020	Real insight into concurrency scheduling	Does not analyse end-to-end ingestion performance	Supports testing reserved concurrency
Wang et al., 2018	Strong cold-start empirical evidence	No ingestion validation	Justifies contextualised testing
Baldini et al., 2017	Establishes memory–latency relationship	Evaluates memory in isolation	Motivates multi-factor testing

Table 2. Critical Summary of Prior Research

3. Research Methodology

3.1 Rationale for Experimental Design

In the present study, AWS Lambda ingestion performance was investigated using a three-factor full factorial design to examine how the interaction of memory allocation, batch/partition size, and reserved concurrency affects execution. Here, a full factorial is used because this approach can capture interactions among factors; for example, high memory with low concurrency still produces longer execution times if CPU capacity is saturated. Observational studies cannot control external variability, and fractional designs run the risk of missing important interactions. Thus, a full factorial design has the highest internal validity and ensures that all three factors are

tested for all their possible combinations, in keeping with well-established directives for research methodology.

Parameter	Levels (Event Workload)	Levels (Batch Workload)	Parameter
Memory Allocation	512 MB, 1024 MB, 2048 MB	512 MB, 1024 MB, 2048 MB	Memory Allocation
Batching / Part Size	Batch sizes: 1, 10, 100	Part sizes: 8 MB, 32 MB, 64 MB	Batching / Part Size
Reserved Concurrency	0, 10, 30	0, 10, 30	Reserved Concurrency

Table 3. Parameter Settings Used for Memory, Batch/Part Size, and Reserved Concurrency

3.2 Hypotheses

The hypotheses to be tested for each experimental factor are defined below in order to facilitate the comprehension of the statistical analyses.

Memory Allocation:

- **H0 (Null) :** Memory allocation does not significantly affect p95 latency, cold-start duration, throughput stability, or cost-efficiency.
- **H1 (Alternative) :** Memory allocation significantly affects at least one of the performance measures listed.

Batch/Part Size:

- **H0 (Null) :** Batch or part size has little significant effect on ingestion latency, throughput stability, or cost-efficiency.
- **H1 (Alternative) :** Batch or part size has significant impact on at least one quantified performance metric.

Reserved Concurrency:

- **H0 (Null) :** Reserved concurrency does not heavily influence ingestion latency nor the stability of throughput.
- **H1 (Alternative) :** Reserved concurrency has a significant impact on at least one of the performance metrics.

The hypotheses presented in this study logically relate the statistical results to the research question and the stated objectives.

3.3 Selected Metrics and Their Justification

In all, four dependent variables were chosen to capture ingestion-specific performance:

- p95 latency is the maximum time it takes to process a message; hence, this is a metric of both the freshness of data and responsiveness of the pipeline to data ingestion.
- Cold-start durations must be quantified; therefore, a historical analysis will be done of cold-start times under different configuration choices to find the effect these configurations have on the overhead of initialization and whether there are any trends emerging from short-duration function executions within the workload.
- By properly assessing throughput variability levels due to variations in execution time, the number of throttling incidents that occurred, and how reserved concurrency corresponds to the rate of throttling incidents that occurred, we can better understand throughput variability. In addition, this analysis will also provide insight on whether increasing execution times and/or concurrency restrictions stabilises throughputs.
- The cost and cost-efficiency are quantified using AWS GB-second billing to articulate the trade-offs between accelerated execution achievable with higher memory tiers and the corresponding increase in per millisecond cost.

As a whole, the metrics above provide answers to the research objectives concerning latency, stability, cold start behaviour, and cost-effectiveness.

3.4 Experimental Constraints

I carried out all the experiments in a tightly controlled environment to ensure internal validity would remain intact while improving reproducibility. More specifically, all testing was done against a single AWS region, eu-west-1, to avoid regional variability; a single Python runtime was used for the purpose of achieving repeatable cold-start behavior across tests. Synthetic workloads are designed with a focus on repeatability to minimize the amount of noise exhibited in production environments. Individual configurations were each deployed as an isolated Terraform module to prevent cross-contamination between configurations. Supporting AWS services (for example, Lambda, S3, CloudWatch) were also assumed to behave normally. Therefore, performance differences observed within these constraints are explained by the experimental variables.

3.5 Statistical Techniques

The effects of memory, batch/part size, and reserved concurrency were analyzed using one-way ANOVA, which allowed for the simultaneous consideration of main effects between the 54 configurations. Significant effects were followed using the Tukey's HSD to identify at which configuration levels the differences occurred. Normality, homogeneity of variance, and independence were assessed via QQ plots/QQ plot comparisons for the normality of each sample, Levene's test for homogeneity of variance for each factor, and lack of any dependent relationship to other factors via factor isolation techniques. Since the residuals followed a normal distribution with equal variances across all configurations, it is appropriate to conduct a one-way ANOVA to determine the significant effects of each configuration on the outcome variable of interest.

4. Design Specification

4.1 High-Level Architecture

A framework that consists of three components was created so that all testing of the configuration of AWS Lambda would be done under controlled conditions and be repeatable, therefore producing reliable data.

- Amazon Web Services ("AWS") Lambda provides numerous variants ("lambdas"), configuration types, batch sizes, part sizes, and reserved concurrency levels to fit many different scenarios. The architecture of AWS lambda allows for both event-driven and batch-style workload ingestion.
- The Storage Layer consists of all functionality implemented within a single, dedicated Amazon S3 bucket, which is kept under steady conditions to ensure predictable data-path behavior.
- This layer uses CloudWatch Logs and Metrics to record the execution duration, 95th percentile latency, cold-start times, concurrency patterns, and cost telemetry.

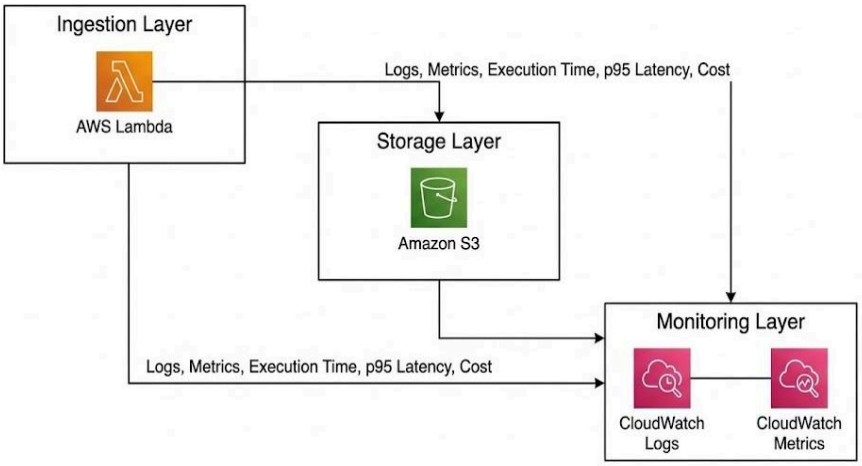

Figure 1. High-level architecture.

4.2 Event-Driven and Batch Ingestion Design

It uses event-driven and batch ingestion strategies for capturing prevailing patterns in serverless data intake. In particular, event-driven ingestion performs data transfers immediately with respect to S3 triggers, whereas batch ingestion orchestrates large sets of file transfers using multipart uploads. Previous work has shown that both batch size and part size can impact both throughput and latency; hence, benchmarking both modes yields a more realistic characterization than would be obtained from a single-workload perspective. Event-driven models are associated with low-latency transfer (Baldini et al., 2017; Jonas et al., 2019), and batch ingestion reflects established behavior for performance in multipart uploads (Winzinger et al., 2021). These design choices align with documented serverless performance principles, where memory–CPU scaling affects execution speed (McGrath & Brenner, 2017) and multipart upload increases throughput for large objects (AWS, 2023).

4.3 Justification for Design Choices

Event-Driven vs Batch Ingestion:

Indeed, the trends in the serverless literature are that event-driven ingestion lowers latency and multipart uploads increase throughput for larger files. However, previous works typically investigate one or the other. This work investigates both together with the aim of explaining cross-factor interactions in real-world ingestion workloads.

Multi-factor Configuration Controls:

While memory, batching behavior, and concurrency have each been explored in isolation, their combined effects remain unexamined within ingestion-specific workloads. The full factorial design here measures these factors together in order to understand how configuration variations affect latency, cold-start behavior, and cost efficiency.

4.4 Configuration Isolation

To provision separate AWS Lambda Functions for each configuration, Infrastructure as Code was leveraged so the test configuration could not be influenced by shared settings, caches, or warm starts. Isolating the test instances means that any differences in performance can be attributed solely to the evaluated Memory, Batch/Partition Size, or Concurrency settings — and that these tests can be exactly repeated across trials.

4.5 Experimental Harness Purpose

Intended to be a testing harness as opposed to an ingestion pipeline used for production; its intention is to produce defined rates of controlled ingest workload, generate repeatable performance measurement and supply the source data to statistical analysis in a 'clean' form, production based criteria that might inform scrutiny around resiliency, error management, or securing, etc., of functions can be relaxed - or removed - to retain clarity of purpose and control over intensity, so that concise but exhaustively defined tests can be run one after another.

4.6 Telemetry and Measurement Design

The telemetry logging system logs derived metadata relating to the behaviour of the ingestion, for most favourable and important metrics touch-points, such as end-to-end latency, and execution time experienced (i.e. time spent ‘ingested’ processing messages), cold/warm start times, batch characteristics, multipart upload behaviour, volume of data successfully ingested, errors encountered, throttling observed, etc. It also logs supplementary compute/throughput detail (including Lambda Source ARN in the AWS billing) that is pertinent to the costs aspect. Telemetry provides a very basic level of help to cost optimised/ingestion process questions in telling how configuration decisions make the performance/cost based metrics appear on computer or manually through the logging of impact by itself, through systems or programmatic algorithms.

4.7 Summary

The architecture provides a reusable template for describing how the various settings and intake mechanisms can be controlled through different patterns of serverless ingestion. It further

describes how each setting-as measured across multiple rounds of execution, bounded workloads, and full telemetry capture-provides an exact knowledge of how these settings impact the cost and performance of the serverless intake pipelines.

5. Implementation

The following chapter describes a serverless data ingestion solution, developed with AWS Lambda and used to execute a factorial analysis of 54 different configurations based on three values per dimension each, or $3 \times 3 \times 3$. Each configuration is able to operate independently of one another and continues to monitor both the performance and configuration of each arrangement over the entire time that they are being processed.

5.1 Lambda Function Architecture

All 54 experimental functions were obtained from a single parameterized Lambda template by systematic variation of three configuration parameters: memory allocation (512, 1024, or 2048 MB), batch or part size (1/10/100 events or 8/32/64 MB parts), and reserved concurrency (0, 10, or 30). This template ensures that only configuration states differ between versions, hence ensuring valid inter-version comparisons. Each function implements deterministic ingestion logic for identifying cold-start conditions, synthesizing input objects, and performing either single-object uploads or multipart uploads while also emitting structured runtime metadata. The design guarantees execution consistency and makes the evaluation of configurations in a controlled framework possible.

5.2 Logging and Telemetry Strategy

Lambda function invocations result in the generation of structured JSON records in CloudWatch Logs at the invocation time. A typical structured JSON record contains:

1. Memory tier, part or batch size and reserved concurrency
2. Start and end timestamps of the invocation
3. Invocation ID and request ID
4. Duration of the execution and the P95 latency (longest 95th percentile latency for the run)
5. Part count (if the invocation was a multipart operation), file upload size and S3 requests made while uploading the file
6. Cold start flag
7. Any errors or retries that were encountered

The JSON payload of each invocation will have a simple identifier that is prefixed by a unique character for easier filtering when performing an extract operation. All 54 configurations are based on the same base record, which provides a consistent and replicable framework for creating a standardized Lambda invocation telemetry. Therefore, there will not be an unfair advantage to creating tons of records of Lambda invocations as compared to its current volume of telemetry.

5.2.1 Cold-Start Identification

The new method to identify cold start events is through a simple, high-performing system of tracking the global "cold start" flag for the first time a Lambda function container is invoked. When a Lambda function's handler exits prior to the replacement of the cold start global flag, that invocation is deemed a cold start. When warm-starting a Lambda function, the execution environment is reused across all prior executions of the function. Thus, no overhead is introduced and all configurations can accurately differentiate between cold and warm-start events.

5.3 Deployment and Environment Setup

Infrastructure provisioning used Terraform and was triggered through an automated CloudShell deployment script to ensure consistency across environments. This deployment workflow included decommissioning the previous resources, instantiating all 54 functions, configuring supporting monitoring components, attaching a predefined IAM execution role, and outputting validation-related information. Performing the deployment in CloudShell provided tooling, credential, and state isolation that was coherent with experimental repeatability.

5.4 Workload Execution Logic

Two ingestion strategies were utilized that represent common serverless data-transfer patterns. The ingestion of events was accomplished by uploading to S3 an entire set of synthetic objects that had been created in one of three different batch sizes (1, 10, or 100). To test the ingested-data processing capabilities of the system by simulating near real-time ingestion, a large number of synthetic objects were created and separated into smaller sections of 8, 32, or 64 MB for upload via the S3 multipart API. In addition to creating these large synthetic objects in a batch, the ETags returned after performing the multipart upload were checked against the original data set to ensure no object had been lost or corrupted in transit. For both batch and individual ingestion, synthetic objects were created deterministically with time control to limit the variability of results not due to differences in configuration.

5.5 Trial Execution Framework

Through the use of a custom experimental runner, 500 executions of 54 configuration sets were performed over a period of five days. The total number of executions in this study was 27000. The use of spaced running with cold start state cleanups resulted in the generation of cold start time periods for some runs. Each run had a unique identifier and timestamps that provide traceability for the dataset. The telemetry information collected for Invocation Level Test Runs is structured and archived based on the structure prefixes used while generating them. Grouping the telemetry information using these structure prefixes facilitates easy collection of the range of telemetry information that should be processed later on.

5.6 Telemetry Capture and Metric Extraction

The dataset used for this work was extracted from CloudWatch logs; the metrics included P95 and average latency, cold starts, a number of calls per function, gigabyte-seconds used by each function, memory used per function, the amount of requests to S3, and the amount of data

scanned per function. In order to find out the cost of every configuration, the pricing provided by AWS through its API was used. All the extracted data was stored in CSV files and JSON objects to make analysis easier and also to perform statistical hypothesis testing on the obtained data.

5.7 Configuration Validation

The integrity of the experiment was kept intact by using several levels of validation. For example, the configuration state was logged at the time of each experiment invocation, runtime segmentation checks during execution were performed to verify correct execution of batch and multi-part executions, and Terraform outputs were validated against the factorial matrix. Manual sampling was used to verify that the levels of concurrency executed, the segmentation methods followed, and the workload selections made were within the ranges of each parameter set forth in the experiment. All these methods provided additional verification that the observed results within the experiment were caused by the experimental variables and not by execution drift.

5.7.1 Idempotency and Ingestion Safety

The key identifier returned for each call was distinct in order to prevent previously uploaded content from getting overwritten while multipart uploads contained ETag verification to ensure that any partial or duplicate uploads were not accidentally committed. When compared to other more resource-intensive methods of transferring data used in production environments, this solution provided appropriate safeguards for experimental control and repeatable runs via definitive telemetry.

5.7.2 Engineering Challenges and Resolutions

In order to create a dependable and stable application, multiple engineering issues were successfully addressed during the application’s development process. In particular, multipart uploads created time bounding constraints that could intermittently terminate uploads. Retry logic was introduced based on S3 API semantics as a means of mitigating this issue. Therefore, the need for the mitigation of temporal state conflicts in Terraform's state was handled by using forced refresh and automated clean-up of temporary files to identify the root cause of the conflict. Inconsistent detection of cold starts when executing AWS Lambda functions was handled by improving the EMF-based classification of cold-starts and warm invocations. Lastly, jitter due to excess concurrency limits on CloudShell invocations was reduced by implementing scheduling controls that limited the maximum level of concurrency. Collectively, these improvements enhanced the data-set’s reliability and made execution conditions more stable.

5.8 Error Handling and Reliability

A combination of components is included in the System to help maintain continuity across Trials. These components include a mechanism to identify failures in IAM Permissions, a mechanism for confirming availability of Log and Storage Resources, methods of handling Multipart Aborts, the ability to monitor Event Throttling and the ability for Technicians to manually invoke some processes to facilitate Debugging. These components all work together to produce Reliable Systems that perform Consistently and collect Complete Data.

5.9 Summary

The automated and repeatable experimental harness system was able to deploy 54 isolated Lambda configurations while ensuring that the ingestion workloads were deterministic as defined earlier (in accordance with requirements) and that all telemetry data could be captured by the deployed system at a finer granularity level, enabling robust statistical evaluation of the performance. This design was strongly internally valid and could directly meet the methodological objectives of this study, including isolating configuration effects, reducing noise, and allowing for the empirical assessment of performance and cost.

Component	**Purpose**	**Notes**
AWS Lambda	Executes event-driven and batch ingestion functions	54 parameterised configurations
Amazon S3	Destination storage for uploaded objects/parts	Used for event and multipart uploads
CloudWatch Logs & Metrics	Captures structured logs and EMF metrics	Used for latency, cold-start, and cost-derived metrics
Terraform	Automates deployment of all infrastructure resources	Ensures configuration isolation
Experiment Scripts	Orchestrate invocations and collect telemetry	Used for trials, run IDs, and metric extraction

Table 4. Summary of experimental system components.

6. Evaluation and Results

This chapter presents the performance and cost outcomes from the 3×3×3 factorial evaluation of AWS Lambda configurations. A total of 27,000 invocations were analysed to assess p95 latency, execution duration, cold-start behaviour, and cost. Results are presented first, followed by interpretation and cross-metric observations.

6.1 Performance Results

6.1.1 p95 and Average Duration

Memory allocation had the most significant effect on ingestion performance. Mean p95 durations for batch ingestion were:

Memory (MB)	**Mean p95 (ms)**
512	42,095.7
1024	21,543.3

2048	12,991.8

Table 5. Mean p95 Duration by Memory Group

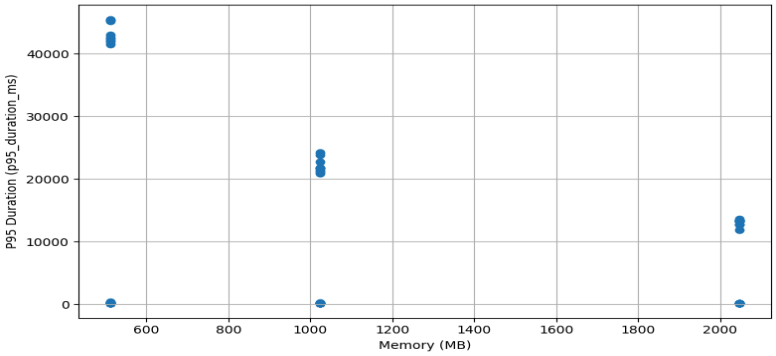

Figure 2. Memory vs P95 Duration

The trend shows a clear and substantial reduction in p95 latency at higher memory tiers. A one-way ANOVA confirmed statistically significant differences:

$F = 4.9005$, $p = 0.0113$

Comparison	Mean Diff	p-value	Significant
512 vs 1024	–10,578	0.0939	No
512 vs 2048	–15,160	0.0099	Yes
1024 vs 2048	–4,582	0.6287	No

Table 5. Tukey HSD pairwise comparisons by memory tier

Tukey HSD results showed that the major significant difference occurred between 512 MB and 2048 MB, while improvements from 1024 MB to 2048 MB were not statistically significant. Given the strong dominance of memory effects observed, a one-way ANOVA was determined to be sufficient for validating the primary performance driver.

6.1.2 Variability Analysis

Variance patterns followed the same trend:

- **512 MB** showed the highest variability
- **2048 MB** showed consistently lower variance
- Event-driven workloads had lower variance due to shorter paths

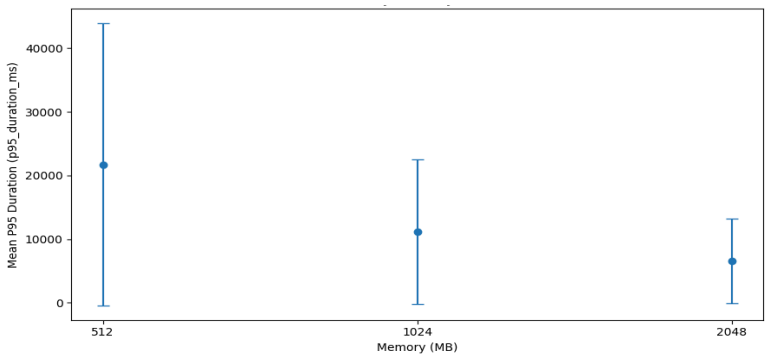

Figure 3. Mean P95 duration by memory tier with standard error bars

6.2 Cold Start Performance

Cold-start behaviour was uniform across all 54 configurations. Average initialisation times were:

Memory (MB)	**Avg Init (s)**
512	0.39s
1024	0.39s
2048	0.38s

Table 6. Average Init Duration Across All Configurations

All cold starts fell within 0.33–0.47 s, with no measurable relationship to memory, batch size, or reserved concurrency. This suggests that cold-start costs were dominated by Python runtime initialisation rather than ingestion workload logic.

6.3 Cost Results

6.3.1 Cost per Invocation

Costs were derived using GB-seconds and S3 request pricing. Average invocation costs were:

Memory (MB)	**Mean Cost (USD)**
512	0.000281
1024	0.000293
2048	0.000337

Table 7. Mean Cost per Invocation

Although the per-millisecond rate increases with memory, execution time decreases significantly, reducing total GB-seconds.

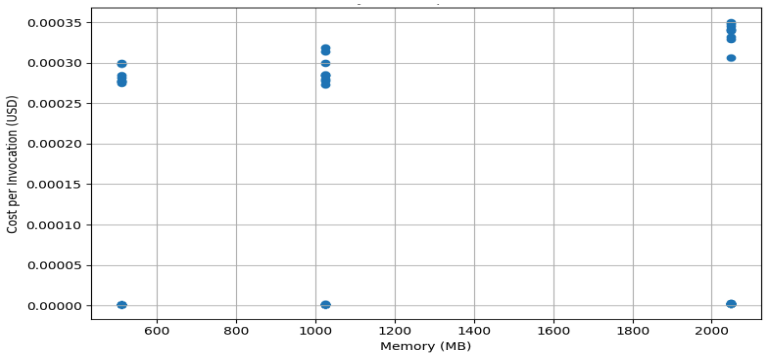

Figure 4. Memory vs Cost per Invocation.

6.3.2 Interpretation

Higher-memory configurations yielded the best cost–performance ratio because their reduced execution time more than offset their higher price per millisecond. For ingestion-heavy workloads, 2048 MB was consistently the most cost-efficient configuration. This cost–performance behaviour is consistent with findings from Hosseinzadeh et al. (2021) and recent AWS performance studies, which similarly note that higher-memory Lambda configurations often reduce total cost due to significantly shorter execution times. While 2048 MB has the highest absolute cost per invocation, the 3.2x reduction in execution duration offsets the 4x memory cost increase, making it the most efficient in terms of cost per unit of work performance.

6.4 Effects of Batch/Part Size and Reserved Concurrency

6.4.1 Batch / Part Size

Batch and part size had moderate effects:

- Small event batches increased S3 PUT overhead
- Larger batches reduced request volume but increased per-invocation duration
- For multipart uploads, 8 MB parts produced the highest overhead, while 32–64 MB parts reduced request count but increased per-part duration

These effects mainly influenced S3 behaviour, not Lambda compute time.

6.4.2 Reserved Concurrency

Reserved concurrency produced weak but observable effects:

- Slight improvement in jitter and throughput stability
- No significant impact on p95 latency or cold-start duration

Concurrency influenced queueing rather than execution.

6.5 Cross-Metric Observations

Correlation analysis showed:

- Strong negative correlation between memory size and p95 latency
- Strong positive correlation between execution duration and cost
- Weak correlations between batch size and cost
- Minimal correlations involving reserved concurrency

Execution duration was therefore the primary determinant of cost efficiency.

6.6 Summary of Findings

1. Memory allocation was the dominant performance factor, showing statistically significant improvements in p95 latency.
2. Cold-start behaviour remained stable across all configurations.
3. Batch/part size had moderate effects, primarily influencing S3 request patterns.
4. Reserved concurrency had weak effects, mostly reducing jitter.
5. 2048 MB provided the best cost–performance outcome(lowest cost per unit of work), with the lowest GB-seconds consumption.

Memory selection is the major contributor towards achieving maximum performance/cost effectiveness in workloads where there are a lot of ingested items in AWS Lambda functions. Batching and Concurrency are the next best way of improving performance and reducing costs.

7. Discussion

This chapter provides a view of the experimental data from the perspective of the research goals and previous work, demonstrating how configuration parameters impact ingestion performance and defining the mechanism through which those parameters exerted their influence on the results.

7.1 Linking Findings to Research Objectives

This research examined the impact of various adjustments made to the memory allocation, batch/partition sizes, and reserved concurrency settings to find how each of these changes influences the P95 latency, throughputs, cooldown times, as well as the costs associated with each. Across both latency and execution time, memory allocation consistently provided the greatest improvements. Batch and part size introduced moderate overhead because of S3 request behaviour, while reserved concurrency offered only minor stabilisation. Cold-start duration remained constant across all configurations. Taken together, these findings fulfil Objectives 1–5 and directly answer the primary research question by demonstrating, with statistical evidence, how the three configuration factors shape latency, cold-start behaviour, throughput stability, and cost in ingestion-heavy Lambda workloads.

7.2 Why Memory Allocation Dominates Performance

Memory allocation emerged as the dominant factor because Lambda increases CPU share alongside memory. Higher memory tiers therefore reduce CPU bottlenecks and shorten I/O wait

times during S3 uploads. The substantial improvement from 512 MB to 2048 MB agrees with earlier work showing that mixed compute–I/O workloads benefit significantly from increased CPU availability.

7.3 Interpretation of Batch and Part Size Effects

Batch and part size had clear but moderate effects. Larger batches reduced S3 control-plane operations but lengthened invocation duration, while multipart uploads showed that small parts add overhead and larger parts reduce request count at the cost of slightly longer transfers. These results confirm that batching primarily affects S3 request overhead rather than Lambda computation, consistent with serverless storage literature.

7.4 Weak Influence of Reserved Concurrency

Reserved concurrency resulted in modest jitter reductions and contributed to more stable throughput, with negligible impact on p95 latency or cold-start durations. This confirms earlier observations that the configuration of concurrency controls scheduling behavior more so than execution velocity, and thus remains beneficial for workloads where predictable throughput is more important than latency minimization.

7.5 Practical Implications for AWS Architects

The study provides several recommendations for ingestion-heavy serverless pipelines:

- First, higher memory tiers are preferable since 2048 MB showed the best performance throughout and is just as cost-effective.
- Second, batch and part size should be optimized to match the behavior of S3, using moderate-to-large part sizes in the range of 32-64 MB along with balanced event batch sizes to reduce request overhead.
- Third, reserved concurrency should be used in a discriminatory manner to stabilize throughput, as it does not provide any reduction in latency.
- Fourth, mitigation of cold-start is relatively less important, since cold-start time was small and consistent with respect to the total duration of ingestion.

Put together, these are a practical configuration decision matrix for tuning AWS Lambda-driven ingestion workloads.

7.6 Methodological and Environmental Limitations

Results are limited by a number of factors: confinement to a single AWS region and runtime, dependence on synthetic workloads, controlled invocation rates, and standardized S3 configurations. While these mitigate generalisability, they strengthen internal validity and allow unambiguous attribution of observed effects to the configuration parameters.

7.7 Summary

Memory allocation is the primary lever for optimising ingestion performance, whereas batch/part size offers secondary opportunities for optimisation and reserved concurrency has limited benefits for stabilisation. Cold-start behaviour was consistent across all configurations. These results align with recent serverless research and offer practical, evidence-based recommendations for the configuration of Lambda ingestion pipelines. These findings fully meet the specified research objectives and precisely answer the central research question.

8. Conclusion and Future Work

This paper investigates how memory allocation, batch/part size, and reserved concurrency interact with each other, affecting performance and cost efficiency in ingestion pipelines running on AWS Lambda. Using a controlled 3×3×3 factorial design, consisting of 54 configurations and 27,000 invocations, the experiment isolates main effects of these parameters on p95 latency, cold-start duration, throughput stability, and compute cost.

Indeed, the results show that ingestion performance is controlled by memory allocation: increasing available memory significantly reduces p95 latency and execution time. Despite a much higher per-millisecond billing rate, the 2048 MB tier proves most cost-efficient for most workloads due to reduced runtime. The injected size of the batch and the other parameters constituting it, have dominant effects on ingestion not because of the computation on lambda, but instead on account of the overhead of requests to S3. Larger size parts and generally equal numbers of requests to an ingestion type will have lower frequency of requests and maintain consistent performance. Using Reserved Concurrency provides only small improvements at best, as it can somewhat reduce jitter and create steadier throughput rates compared to Lambda Without Reserved Concurrency, however, latency and cold start times remain unchanged. Cold-start duration is identical regardless of configuration, which confirms that runtime initialization-not workload characteristics-is the dominant factor that determines cold-start cost in ingestion workloads.

This research is the first statistical method of validating a multi-factor experimental design of AWS Lambda ingestion configurations. The results will provide a reproducible method to conduct future research on the performance of serverless computing as well as offer an easy-to-follow configuration decision matrix for architects and developers in tuning ingestion-heavy serverless pipelines.

Future studies can build upon these findings by examining additional configurations across different AWS regions and managed runtime options using real-world ingestion datasets, integrating upstream services such as Amazon Kinesis and Amazon SQS, and developing adaptive/dynamic configuration methods that automatically optimize cost versus performance as workloads change.

References

Amazon Web Services. (2024). *Serverless Compute Performance Deep Dive (COM401)*. AWS re:Invent 2024. Retrieved from https://aws.amazon.com/blogs/

AWS (2023a) *AWS Lambda performance optimisation documentation*. Available at: https://docs.aws.amazon.com/lambda/latest/dg/configuration-optimization.html

AWS (2023b) *Amazon S3 multipart upload performance guidelines*. Available at: https://docs.aws.amazon.com/AmazonS3/latest/userguide/mpuoverview.html

Baldini, I. et al. (2017) 'Serverless computing: Current trends and open problems', in Raj, P. & Raman, A. (eds.) *Research advances in cloud computing*. Singapore: Springer. Available at: https://doi.org/10.1007/978-981-10-5026-8_1

Bluemke, I. & Zdanowski, A. (2025) 'Evaluation of configurations of AWS Lambda functions', *International Journal of Electronics and Telecommunications*, 71(3). Available at: https://ijet.pl/index.php/ijet/article/view/10.24425-ijet.2025.153619

Bodner, T., Radig, T., Justen, D., Ritter, D. & Rabl, T. (2025) 'An empirical evaluation of serverless cloud infrastructure for large-scale data processing', in *Proceedings of EDBT 2025*. pp. 935–949. Available at: https://doi.org/10.48786/edbt.2025.76

Fouladi, A. et al. (2019) 'From laptop to Lambda: Outsourcing everyday jobs to transient functional containers', in *USENIX Annual Technical Conference (USENIX ATC 19)*. pp. 517–532. Available at: https://www.usenix.org/conference/atc19/presentation/fouladi

Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C., Khandelwal, A., Pu, Q., Venkataraman, S., Ranganathan, P., Stoica, I. & Popa, R. (2019) *Cloud programming simplified: A Berkeley view on serverless computing*. Berkeley, CA: University of California Technical Report No. UCB/EECS-2019-3. Available at: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf

Karamzadeh, A. & Shameli-Sendi, A. (2024) 'Reducing cold start delay in serverless computing using lightweight virtual machines', *Journal of Network and Computer Applications*, 232, 104030. Available at: https://doi.org/10.1016/j.jnca.2024.104030

Ustiugov, D., Skogias, D., Marques, R., & Batista, M. (2021). *Benchmarking, analysis, and optimization of serverless cold-start latency.* Available at: https://ustiugov.github.io/assets/files/REAP_ASPLOS21.pdf

Marcelino, C.A. & Nastic, S. (2024) 'Truffle: Efficient data passing for data-intensive serverless workflows', in *Proceedings of IEEE UCC 2024*. pp. 53–62. Available at: https://doi.org/10.1109/UCC63386.2024.00017

McGrath, G. & Brenner, P.R. (2017) 'Serverless computing: Design, implementation, and performance', in *Proceedings of ICDCS Workshops 2017*. pp. 405–410. Available at: https://doi.org/10.1109/ICDCSW.2017.36

Remala, R., Mudunuru, K.R. & Nagarajan, S.K.S. (2024) 'Optimising data ingestion processes using a serverless framework on Amazon Web Services', *International Journal of Computer Trends and Technology*, 72(6), pp. 118–125. Available at: https://doi.org/10.14445/22312803/IJCTT-V72I6P116

Shahrad, M., Fonseca, R., Goiri, Í., Chaudhry, G., Batum, P., Cooke, J., Laureano, E., Tresness, C., Russinovich, M. & Bianchini, R. (2020) 'Serverless in the wild: Characterising and optimising the serverless workload at a large cloud provider', in *Proceedings of USENIX ATC 2020*. pp. 205–218. Available at: https://www.usenix.org/conference/atc20/presentation/shahrad

Wang, L., Li, M., Zhang, Y., Ristenpart, T. & Swift, M.M. (2018) 'Peeking behind the curtains of serverless platforms', in *Proceedings of USENIX ATC 2018*. pp. 133–146. Available at: https://www.usenix.org/conference/atc18/presentation/wang-liang

Gadban, F., & Kunkel, J. (2021). *Analyzing the performance of the S3 object storage API for HPC workloads*. Applied Sciences, 11(18), 8540. https://doi.org/10.3390/app11188540