National College of Ireland

Optimised Latency in Cloud Systems: Multi-Level Caching through AWS-Based FDN

MSc Research Project
MSCCLOUD

Harish Challa
Student ID: 23417498

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Harish Challa……

Student ID: ……23417498……..……

Programme: ……MSCCLOUD…… **Year:** …2025-26..

Module: …… MSc Cloud Computing Research Project ……..……

Supervisor: …… Shreyas Setlur Arun ……..……

Submission Due Date: ……11 Dec 2025……..……

Project Title: Optimised Latency in Cloud Systems: Multi-Level Caching through AWS-Based FDN

Word Count: ……7128…… **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: ……Harish Challa……

Date: ……10 Dec 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 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):	

Optimised Latency in Cloud Systems: Multi-Level Caching through AWS-Based FDN

Harish Challa

23417498

Abstract

Serverless computing has become a transformative model in cloud architecture by eliminating manual server management and enabling on-demand function execution. However, there are some challenges like cold start latency, repeated computation and stateless execution mostly degrade performance and hinder real-time responsiveness. This study proposes a Function Delivery Network (FDN) framework that extends the principles of Content Delivery Networks (CDNs) to function-level execution by using a multi-level caching mechanism to increase performance. The system uses a Level-1 in-memory Least Recently Used (LRU) cache for instant retrieval and a Level-2 persistent cache using AWS DynamoDB to store preprocessed type of results for minimizing redundant AWS Lambda invocations. This study has implemented through Flask, API Gateway and AWS Lambda within the AWS ecosystem. The FDN is experimentally validated using text and PDF files ranging from 40 KB to 2 MB. This study shows over 95% reduction in response latency, with Lambda executions averaging 492–2638 ms reduced to 0–21 ms through caching layers. These findings confirm that hierarchical caching enhances throughput, scalability, and cost efficiency while reducing cold starts.

Keywords: Serverless Computing, Function Delivery Network, Multi-Level Caching, AWS Lambda, Latency Optimization

1 Introduction

1.1 Background and Motivation

The concept of serverless computing has changed the face of the modern cloud architecture because servers no longer need to be managed manually; they can be invoked at any time on-demand using services such as AWS Lambda (Mehdi Syed and Ali, 2024). Although this model is scalable and cost effective, the latency of cold start, repetitiveness of computation and stateless clouds remains a significant barrier to real time responsiveness. Such latency problems are crucial in applications where rapid processing and rapid access to data is required (Shukla et al., 2023). In order to overcome this drawback, the Function Delivery Network (FDN) concept applies the concepts of Content Delivery Networks (CDNs) to functions through execution at a function level by deploying and caching serverless functions

nearer to the users (Yang et al., 2023). The motivation of this study was the increasing requirement on minimizing latency and enhancing throughput in distributed serverless systems by using smart caching policies. The study will add a multi-level caching system, an in-memory LRU cache to access the data in a short period of time and DynamoDB to store the data permanently, to improve the performance of the executions, reduce unnecessary Lambda invocations, and provide low-latency behaviour under various workloads and file types.

1.2 Importance of the Study

The importance of this study lead to the enhancement of the performance and scalability of a serverless computing by optimized Function Delivery Network (FDN). The research can eliminate one of the key drawbacks of AWS Lambda, namely, the execution latency that is caused by repeated invocations and cold starts by introducing a multi-level caching system based on in-memory LRU and DynamoDB. The paper offers a comprehensive understanding on how hierarchical caching can greatly decrease the response time, improve cost-efficiency as well as facilitating real-time data processing, which renders it very useful in today cloud applications and latency-sensitive applications.

1.3 Research Question

Serverless computing systems often experience performance bottlenecks due to cold starts, repeated function invocations, and stateless execution models that increase overall latency. These issues are particularly evident when processing large or frequently accessed data files in distributed cloud environments. Traditional single-level caching or no-caching approaches fail to efficiently minimize redundant computations and maintain real-time responsiveness. Therefore, this study seeks to investigate how the implementation of a multi-level caching mechanism, integrating an in-memory Least Recently Used (LRU) cache with a persistent DynamoDB layer, can enhance execution performance in a Function Delivery Network (FDN). The research question guiding this investigation is:

"How does a multi-level caching mechanism (in-memory LRU + DynamoDB) impact the function execution time and overall end-to-end response latency compared to single-level and no-caching approaches in an AWS-based Function Delivery Network (FDN)?"

1.4 Research Objectives

There are three research objectives in this study which includes:

1. To design a multi-level caching FDN model (L1 LRU + L2 DynamoDB) to reduce latency and avoid redundant Lambda executions.
2. To evaluate performance across no-cache, single-level, and multi-level caching modes using latency and execution metrics.
3. To build and test a prototype using Flask and AWS services to validate improved speed, scalability, and cost efficiency.

1.5 Structure of the Study

Table 1: Overview of the Study Structure and Chapter Descriptions

Chapter	Description
Chapter 1: Introduction	This chapter introduces the research background, highlights the importance of the study, presents the central research question, outlines the objectives, and explains how the overall study is structured.
Chapter 2: Related Work	This chapter reviews existing literature on serverless computing, latency challenges, multi-level caching mechanisms, Function Delivery Networks (FDNs), and machine-learning-based optimisation techniques relevant to this study.
Chapter 3: Methodology	This chapter explains the research design, outlines the methodological approach, details the AWS-based infrastructure configuration, and describes the performance metrics used for data collection and evaluation.
Chapter 4: Design Specification	This chapter describes the overall system setup, presents the architectural components of the proposed Function Delivery Network (FDN), and explains the multi-level caching workflow designed for latency optimisation.
Chapter 5: Implementation	This chapter demonstrates the practical implementation on AWS, including setup of services, cache integration, and the pseudocode used for processing TXT and PDF files.
Chapter 6: Results	This chapter presents experimental findings, including text and PDF file processing outcomes, execution time comparisons, latency analysis, multi-level cache performance, and comparative evaluation with existing studies.
Chapter 7: Conclusion and Future Work	This chapter summarises the research outcomes, highlights key contributions, discusses limitations, and proposes directions for future enhancements in caching and serverless optimisation.

2 Related Work

2.1 Serverless Computing and Function-as-a-Service (FaaS)

The concept of serverless computing has become one of the revolutionary concepts in cloud computing that enables developers to run application functions without having control over the underlying infrastructure (Wen et al., 2023). This architecture is devoted to event-based execution with which each functionality will be called based on certain triggers and which will be automatically scaled based on demand. A proficiently designed and adopted all-to-a-service (FaaS) framework has been adopted by vendors like AWS Lambda, Microsoft Azure Functions, and Google Cloud Functions to allow optimal resource usage and billing by execution, lowering operational expenses and overheads (Shafiei et al., 2022). The primary benefit of serverless computing noted by researchers is that it can simplify the complexity of the infrastructure, which provides agility, scalability, and elasticity appropriate in modern

cloud-native applications. Nevertheless, research also indicates a high level of difficulties related to cold start latency, statelessness and orchestration of functions, which might affect system responsiveness and general performance. In order to address these problems, the latest methods include pre-warming strategies, caching systems, and adaptive control of functions to reduce the delay of the start-up and preserve consistent states during the execution process. Moreover, developments generated in workload forecasting and distributed caching have put FaaS not just as a responsive architecture but as a self-optimizing architecture. The resultant situation is that serverless computing has become a paradigm towards attaining high performance, cost efficiency, and dynamism in distributed clouds.

2.2 Latency Challenges in Serverless Environments

Despite the ease of automatic scaling and cost effectiveness of serverless computing, one of the most important performance bottlenecks of serverless computing is latency. Latency in serverless environments is mainly caused by cold start problem where the idle use of a given function is invoked after a certain duration of dormancy (Chaudhary et al., 2024). In the given case, the cloud provider should allocate resources, create containers, load dependencies, and start the runtime environment, and this causes delays of several seconds and hundreds of milliseconds (Adeppady et al., 2023). This start-up cost has a devastating effect on time-sensitive and real-time applications like data analytics, data streaming, and content processing.

Besides cold starts, the delay in network transmission among the network components such as API Gateway, Lambda, and storage (e.g., S3 or DynamoDB) are also a source of total response time (Golec et al., 2025). The statelessness of FaaS functions makes the situation worse because every invocation is initialised with a clean environment (Puliafito et al., 2022) and does not retain past context leading to repetitive computation and data retrieval. It has also been found that there is concurrency-related latency when instances of the functions are competing over scarce resources under heavy load conditions. In a bid to reduce the issues, researchers have suggested the following techniques which include: function pre-warming, intelligent scheduling, and multi-level caching mechanisms. These approaches are designed to preserve hot execution and decrease the time of reinitiating and enhance the overall responsiveness of serverless applications in production-scale.

2.3 Multi-Level Caching for Performance Optimization

Multi-level caching is now a vital feature of enhancing the speed of data retrieval, reducing redundant computation, and I/O operations optimization in serverless computing systems (Yu et al., 2024). Multiple-level cache should include two layers, one in-memory (L1) cache with quick and short-term access to data and one persistent distributed (L2) cache like Amazon DynamoDB or Redis with long-term storage and retention. Such hierarchical design enables systems to lessen the reliance on re-executions of the Lamdas by putting prepared outcomes out of caches, which helps to enhance the response time and decrease the operational expenses.

Recent studies have come up with multi-level caching with adaptive and asynchronous optimization. CMCache being an adaptive cross level data placement algorithm proves to be an example whereby the level of the caching and the new data can be managed independently to ensure optimal level of caching. CMCache has been capable of realizing a decrease in the rate of cache misses and averaging a response time of up to 89 and 79 percent respectively by dynamically assigning cache space and switching strategies in response to access patterns (Zeng et al., 2025). Similarly, asynchronous multi-level checkpointing has also been proposed as a method of promoting I/O throughput and access to data in high-performance computing environment. In this technology, local storage levels and GPU memory are used at speeds up to 74 times faster checkpoint and restore throughput and I/O wait times are also cut by 2 times because both frequently accessed data are pre-fetched and cached in-memory (Maurya et al., 2023).

When applied to AWS-based Function Delivery Networks (FDN), adaptive multi-level caching strategies that incorporate both in-memory LRU caching and persistent DynamoDB can be of great benefit in terms of minimizing end-to-end latency, bandwidth consumption, and unnecessary computations. This can be used to serve repeated requests with serverless functions serving the requests almost in real time with high scalability and near real time response. Therefore, multi-level caching will be one of the pillars of the performance optimization and the consistency of the quality of services in the current serverless architectures that use clouds.

2.4 Evolution of Function Delivery Networks (FDN)

The development of Function Delivery Networks (FDN) is a milestone in the distribution of the use of serverless architectures in terms of distributed functions and their optimization of latency (Corporan et al., 2025). Content Delivery Networks (CDNs), built following the design goals of Content Delivery Networks, which store and provide static content more closely to users, provide inspiration behind the FDNs (Tyagi, 2025), which provide computational services and their products in numerous cloud locations, thereby enabling more effective and quicker execution. The introduction of FDNs solve the major shortcomings of the existing serverless models, namely the higher latency caused by centralized execution and the repetitive calls of the same functions. UDP FDNs facilitate the execution using geographical proximity, which is achieved by means of dynamically deploying functions at distributed edge nodes (Carota et al., 2024) and caching the results of the functions at various levels that minimize the network delay and enhance overall responsiveness.

Initial applications put emphasis on regional functional replication, but the current FDNs incorporate intelligent routing, adaptive caching, and workload prediction to have the regularly used functions executed at the closest or most optimal node. This will reduce unnecessary calculations and load balancing of various Lama instances. Besides, the integration of multi level caching systems - comprising of in-memory and persistent data repose - maximizes FDN efficiency by allowing rapid recovery of stored answers. In turn, the

Function Delivery Network paradigm turns serverless computing into a globe-optimal, latency-sensitive network and provides close-to-real performance and cost-effective scaling to data-intensive and distributed applications.

2.5 Machine Learning Integration in Function Optimisation

Recent research has emphasized the application of the Machine Learning (ML) methods to the complex optimization of functions in different fields. In the study (Liu et al., 2024), the topic is Job-Shop Scheduling, in which a Deep Neural Network (DNN) was included in a Surrogate Lagrangian Relaxation (SLR) model to estimate near-optimal solutions to subproblems, which can overcome the problem of learning difficulties of high-part diversity and large-scale optimization. In the same manner, research (Wu et al., 2024) concentrated on Unit Commitment (UC) in power systems, where the predictive subproblem solutions can be efficiently accomplished with the usage of ML-enhanced SLR, multiplier aggregation, tailored loss functions, and a hybrid of offline and online learning to guarantee the overall performance is maintained at a near-optimal level. In renewable energy, study (Ahmad et al., 2024) used gradient boosting algorithms (XGBoost, LGBM, AdaBoost) and Response Surface Methodology (RSM) to develop and predict and optimize the yield of biogas production, and found the optimal operating conditions and tested predictions experimentally. Following that (Chen et al., 2024) came up with a probabilistic caching and probabilistic function orchestration model of serverless edge computing. They employed their system based on LSTM requests prediction, adaptive placing of containers with the use of multiple edge nodes and minimized cold-start delays. It has a similar result on Knative with around three times higher average response time and cold-start frequency compared to baseline algorithms. Lastly, research (Rai et al., 2024) addressed real-time weed detection with aerial images by introducing the YOLO-Spot model, which is an optimized version of YOLOv7-tiny to fold in complexity and improve its accuracy and the ability to use resource-constrained edge computers to implement precision agriculture. All of these findings indicate that the combination of ML and problem-specific optimization models can improve predictive accuracy, computational efficiency, and flexibility, but it is still difficult to manage the model generalization, scalability, and resilience to a variety of operational conditions.

3 Methodology

3.1 Research Design Framework

The study aims at assessing how a multi-level caching mechanism can be used to optimize the time of running functions and minimizing latency time in a serverless cloud network, which is called the Function Delivery Network (FDN). The research follows the CRISP-DM model, which consists of business insight, data preparation, modeling, assessment, and implementation. The FDN model uses the AWS Lambda with API Gateway as a connection to Flask-based backend and a two-level caching architecture with an in-memory Least Recently Used (LRU) cache (Level 1) and AWS DynamoDB (Level 2). The design will

enable the comparison of no-cache and single-level and multi-level cache cases to ascertain the influence on the response latency and execution efficiency. The use of AWS services i.e. S3, EC2 and Cloud9 help to create realistic serverless setups. The framework provides a systematic replicable process of examining performance gains with caching realization in distributed, on-demand systems of functions delivery.

3.2 Research Methodology Approach

The study is an experimental and quantitative research which assumes the use of latency and performance as a gauge of success in a serverless computing environment. Its methodology involves test of the Function Delivery Network (FDN) through experiment testing in three conditions of caching namely, no cache, single-level cache and multi-level cache. Quantitative data such as the execution time, latency and percentage of hits on the caches are used to measure the performance of the caching layers. The study is conducted on controlled experimental basis where identical inputs and workloads are used in settings to provide fairness and comparisons. It is possible to use this empirical approach in order to assess performance measures in such a way that can be replicated and measured to establish a data-driven foundation defining the impact of multi-level caching on the performance of functions and response time in the distributed serverless setting.

3.3 AWS Infrastructure Configuration

The suggested Function Delivery Network (FDN) architecture is implemented on the AWS ecosystem as a whole to guarantee scalability, elasticity, and without any discontinuity between compute and storage modules. The architecture does have installation of interrelated services that jointly handle calculations, caching, and delivery of data. Every component will be configured to simulate the real-life interaction of cloud-based serverless, in which client requests pass through several layers of caching before it can reach the execution endpoint. This IAM roles and policies are set in place to ensure access security and service-level permissions of all the resources. The infrastructure configuration can be employed to achieve experimental conditions as well as the validation of latency at the production scale. The following table 2 provides a detailed overview of the AWS infrastructure components and their respective roles within the experiment:

Table 2: AWS Infrastructure Components and Their Functional Roles

AWS Component	Purpose / Functionality	Configuration Details
IAM Roles and Policies	Defines secure access permissions for Lambda, API Gateway, and DynamoDB.	Roles created for Lambda execution and DynamoDB access with least-privilege principles.
Amazon EC2 Instance	Hosts the Flask web server and in-memory Level-1 cache (LRU).	Configured with Ubuntu 22.04, Python 3.9, Flask, and Redis (for LRU caching).

AWS Cloud9 Environment	Provides integrated development and deployment workspace.	Used to develop, test, and deploy Lambda functions and APIs.
AWS Lambda Functions	Executes file-processing logic on-demand for text and PDF files.	Two functions created for separate file types; integrated with API Gateway endpoints.
Amazon API Gateway	Acts as an interface between the client and serverless functions.	Configured with REST endpoints triggering Lambda via HTTPS.
Amazon S3 Bucket	Stores uploaded files and Lambda outputs for further processing.	Versioning and access control enabled; lifecycle policy applied for storage efficiency.
Amazon DynamoDB	Serves as Level-2 cache and persistent data store for processed results.	Table includes primary key (file hash), response payload, and timestamp.
Amazon CloudWatch	Monitors performance metrics, latency logs, and Lambda execution time.	CloudWatch dashboards configured for experiment data collection.

3.4 Metrics and Data Collection

In order to measure the performance of the proposed Function Delivery Network (FDN), the following important metrics were measured, namely execution time, response latency and the ratio of hits on the cache. Execution time measures the time taken in AWS Lambda processing and response latency the total end to end delay to the client. The performance of the cache was measured by capturing the response served by Level 1 (in-memory LRU) and Level 2 (DynamoDB) caches in comparison to direct execution in Lambda. The data was gathered through the performance of controlled experiments using different file sizes (TXT and PDF) by running the experiment many times to be able to record the results and guarantee consistency and reliability of the observed performance trends as presented in table 3.

Table 3: Performance Metrics and Their Descriptions

Metric	Description
Execution Time (ms)	Time taken by Lambda to process input and return output.
Response Latency (ms)	Time between client request and full response delivery.
Cache Hit Ratio (%)	Ratio of requests served from cache vs. Lambda.

4 Design Specification

4.1 System Setup Environment

The proposed Function Delivery Network (FDN) is implemented using a Python Flask framework as the backend and a lightweight frontend built with HTML, CSS, and JavaScript.

The entire system is deployed on Amazon Web Services (AWS), leveraging cloud-native services for scalability and performance. Core services include AWS Lambda for serverless execution, API Gateway for request routing, DynamoDB for Level-2 caching, and S3 for file storage, with development supported by EC2 and Cloud9. The coding and integration are carried out in Visual Studio Code and AWS Cloud9 as shown in Table 4.

Table 4: System Setup and Environment Components

Component	Description
Backend	Python Flask Framework
Frontend	HTML, CSS, JavaScript
Cloud Platform	Amazon Web Services (AWS)
Services Used	AWS Lambda, API Gateway, DynamoDB, S3, EC2, Cloud9
Development Environment	Visual Studio Code and AWS Cloud9
Testing Data	Text and PDF files of varying sizes (40 KB – 2 MB)

4.2 System Components and Architecture

The proposed Function Delivery Network (FDN) architecture will ensure the optimal performance of serverless computing with a multi-level caching system and this architecture with AWS cloud infrastructure as presented in Figure 1. The system starts with a client who can engage with the Flask-based web application which is created with HTML, CSS and Java Script. In case a user uploads a document file (TXT or PDF) the request is submitted to the AWS API Gateway, which is a secure entry point to invoke services in the back end. The API Gateway will activate AWS Lambda functions to process text, extract data and generate responses without any dedicated servers.

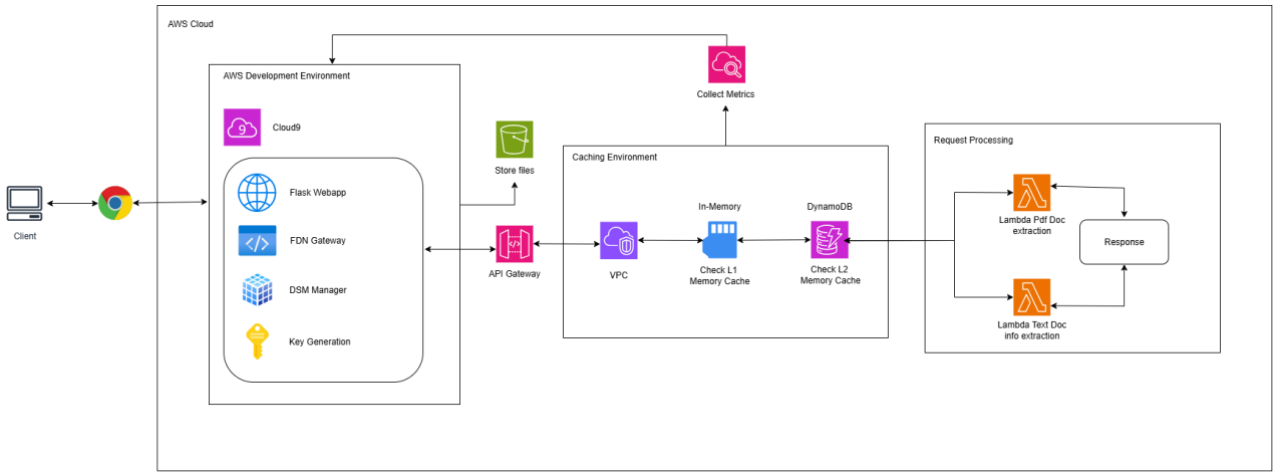

Figure 1: Architecture Diagram

A two tiered caching architecture is implemented to improve performance with Level 1 (L1) being an in-memory Least Recently Used (LRU) cache to provide quicker access to frequently accessed responses and Level 2 (L2) being DynamoDB to provide persistent and scalable data storage. In case there is a requested response in the L1 cache the response is

immediately sent back to the system; whereby the system then searches L2 and lastly executes the request via Lambda in case both caches are not available. Development and deployment are done using AWS Cloud9 and EC2 and the files uploaded are stored by S3. This architecture is effective in minimizing latency, enhancing throughput, and making use of cost-efficient and scalable execution of functions in the cloud environment that are distributed.

4.3 Research Specification: Multi-Level Caching and Latency Optimization

Figure 8 shows System Data Flow illustrates the operational logic of the Function Delivery Network (FDN) designed to optimize serverless performance through a multi-level caching mechanism. When a client uploads a TXT or PDF file through the Flask web app, a unique hash key is generated from the file's content. The system first checks the L1 cache (In-Memory LRU) for the corresponding result; if found, it is instantly returned, minimizing latency. If the L1 cache misses, the system queries the L2 cache (AWS DynamoDB) for previously stored outputs. In case of another miss, the request is forwarded to AWS Lambda, which processes the file and extracts text. The output is then written back to both cache levels for faster future access and returned to the client along with metrics. Finally, performance indicators such as execution time, cache hit rate, and end-to-end response latency are logged for analysis.

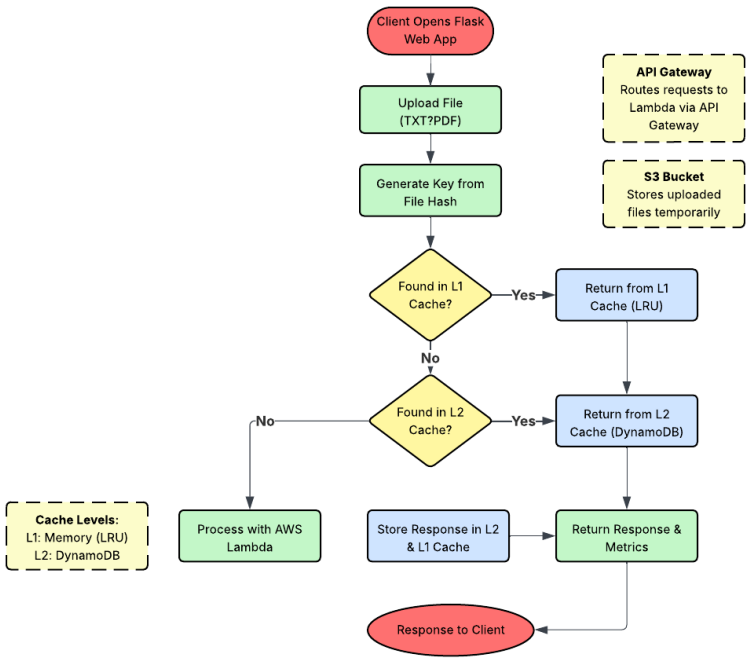

Figure 2: System Data Flow

5 Implementation

5.1 AWS Setup

In Figure 3, the use of AWS DynamoDB as the Level-2 cache (L2) in the Function Delivery Network (FDN) can be observed. In case, the client uploads a file and the reply is not present in the in-memory LRU cache system makes a check with DynamoDB to see whether a result has been previously processed. Key-value pairs are stored in the table named DSMTable and the key is the hash of the file and the value will contain the extracted value. This minimizes the unnecessary Lambda calls and makes the general latency minimal.

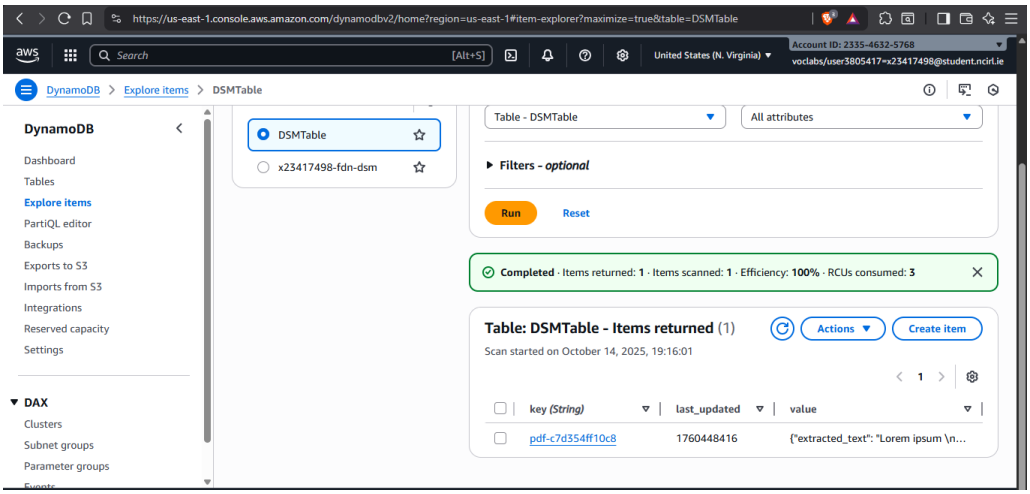

Figure 3: AWS setup: DynamoDB

Figure 4 shows the deployment of the AWS Lambda functionality combined with API Gateway in event triggering. The API Gateway is invoked to process a PDF file whenever a client uploads this file through the Flask web interface. The operation retrieves the text content and gives back the response which is then saved on the multi-level cache (L1 in-memory and L2 DynamoDB) to access later on. This serverless architecture removes the issue of having to manage backend servers, allowing scalable and efficient execution and greatly reducing the latency and improving the overall responsiveness of the system in the Function Delivery Network.

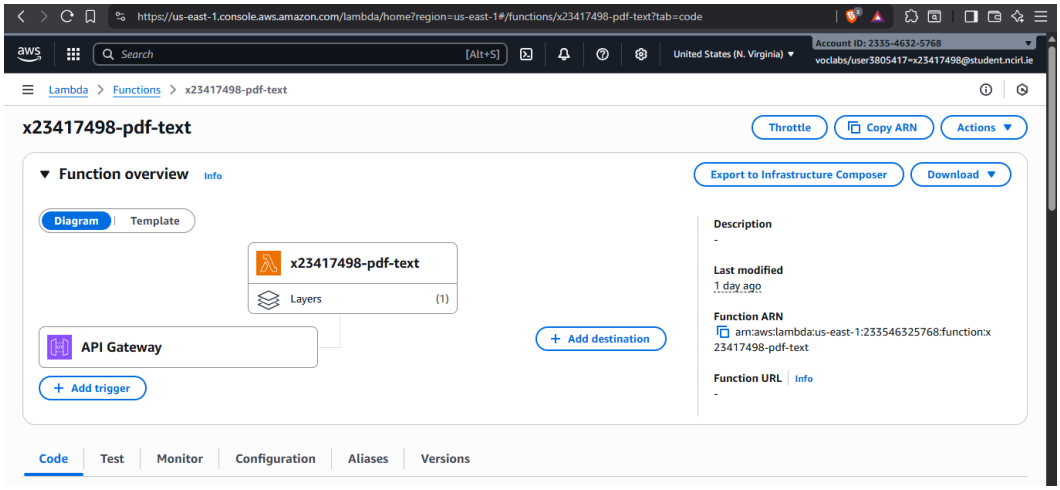

Figure 4: AWS Lambda

5.2 Pseudocode for Txt File

Pseudocode 1: Txt File

Begin
Open Flask Web App
Upload txt_file

text_data ← extract_text(txt_file)
file_key ← generate_hash_key(txt_file)
if file_key IN L1_Cache THEN
    response ← L1_Cache[file_key]
    return response
else if file_key IN L2_Cache (DynamoDB) THEN
    response ← L2_Cache[file_key]
    store response IN L1_Cache
    return response
else
    response ← invoke_AWS_Lambda(text_data)
    store response IN L1_Cache
    store response IN L2_Cache
    return response
end if
collect performance_metrics
end

5.3 Pseudocode for Pdf File

Pseudocode 2: Pdf File

begin

open Flask Web App
upload pdf_file
text_data ← extract_text(pdf_file)
file_key ← generate_hash_key(pdf_file)
if file_key IN L1_Cache THEN
    response ← L1_Cache[file_key]
    return response
else if file_key IN L2_Cache (DynamoDB) THEN
    response ← L2_Cache[file_key]
    store response IN L1_Cache
    return response
else
    response ← invoke_AWS_Lambda(text_data)
    store response IN L1_Cache
    store response IN L2_Cache
    return response
end if
collect performance_metrics
end

6 Results

6.1 Experiment 1: Text File Processing

Figure 5 illustrates the implementation of the Function Delivery Network (FDN) web interface during Experiment 1, where a text file is uploaded and processed through AWS Lambda. The Flask-based frontend enables the user to select and upload a TXT file, which is then assigned a unique cache key. Since the request results in a cache miss, the file is forwarded to the AWS Lambda function for text extraction and processing. The results, including response time and source (Lambda), are displayed on the dashboard, showcasing the integration of caching, API Gateway, and cloud-based computation.

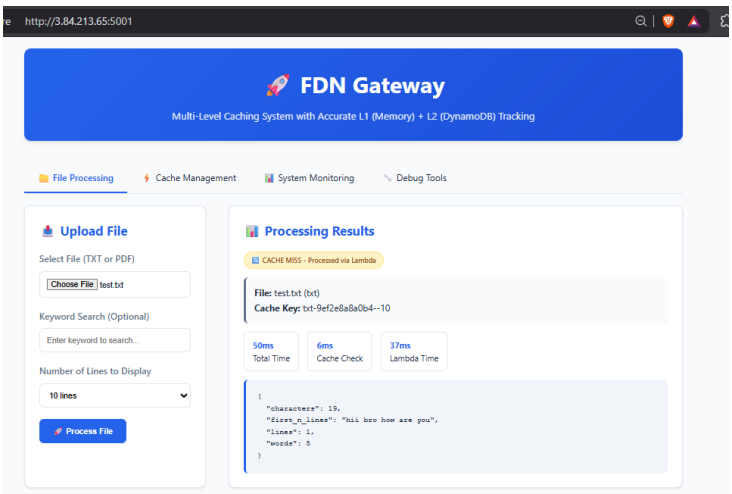

Figure 5: TXT File Processing via Lambda Function

Figure 6 shows the FDN Gateway interface when the second request of Experiment 1 is made whereby the same TXT file is reprocessed. Because the unique cache key of the file is already present in the L1 cache, the system will also receive a result only out of the in-memory cache without calling the AWS Lambda function. This shows the performance of the Least Recently Used (LRU) caching system deployed in Python in Flask system. The latency is minimized to close to zero milliseconds and the overall performance of the system is optimized with L1 caching, as the response time is minimized, and reduces to near zero milliseconds.

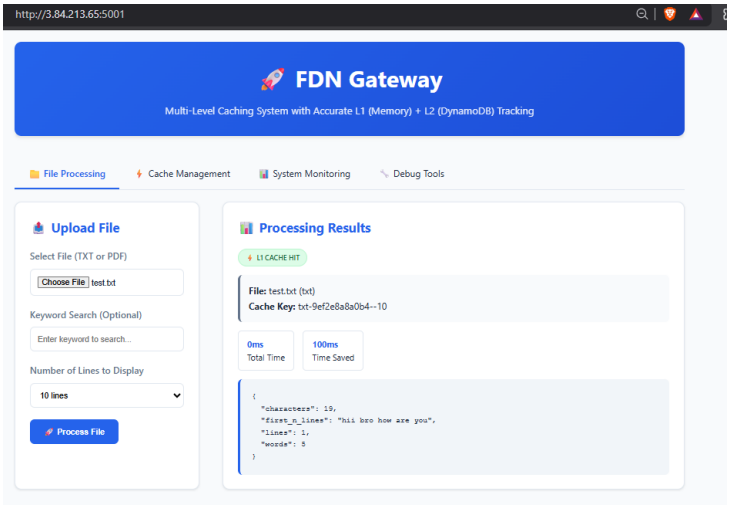

Figure 6: TXT File Response from L1 In-Memory Cache

Figure 7 depicts the cache management interface of FDN Gateway, that is, the L1 cache clearing feature. The system is based on a hierarchical three-level caching (L1, L2, L3) architecture having autonomous control systems. When the Clear L1 Cache button (red button on Cache Controls section) is enabled, the system will perform a flush operation on the first-level cache layer and it will maintain the integrity of the L2 and L3 cache layers. Presented cache metrics have 33.33% cache hit and 1/ 50 entries stored in cache, which means that L1 is not actively used. The operation causes instantary memory de-allocation and recalibration of performance counters, set L1 to new data population and keep the system operational by keeping the cache levels low.

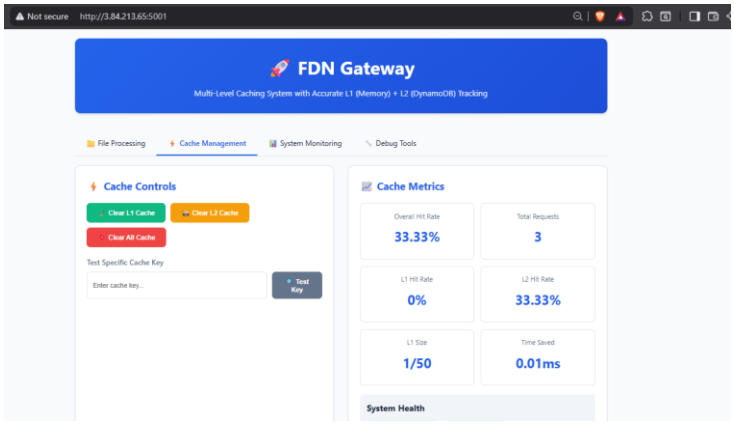

Figure 7: L1 Cache Clearing Operation in FDN Gateway System

Figure 8 shows a gateway cache hierarchy failover of the FDN Gateway after the clearance of the L1 cache. In the case of the requesting of the file: test.txt, following the invalidation of L1 cache, the system implements a hierarchical search strategy. The request initially consults L1 cache, then a cache miss is noticed which automatically escalates to L2 cache which is stored in DynamoDB. This deployment demonstrates the strength of a multi-tier caching where DynamoDB is used as the persistent secondary storage. The answer contains performance indicators 6ms overall time, 6ms cache check time, and 94ms time saved, which proves that L2 retrieval was successful without a backend reprocess, and, thus, the system efficiency did not reduce due to the unavailability of L1.

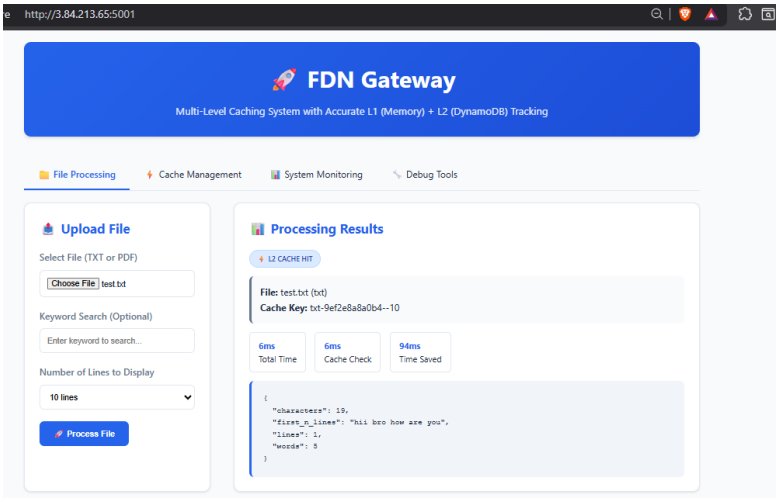

Figure 8: L2 Cache (DynamoDB) Response Operation

6.2 Experiment 2: PDF File Processing

This figure 9 represents the Experiment 2 which demonstrates the capabilities of PDF text extraction with the help of AWS Lambda serverless processing. When a PDF file (sample-cpr.pdf) is uploaded through the File Processing interface, the system bypasses the layers of cache and immediately calls Lambda function to initial processing. The implementation

makes use of serverless architecture in which case the textual contents of the document are extracted with the help of PDF parsing algorithms that are executed by Lambda.

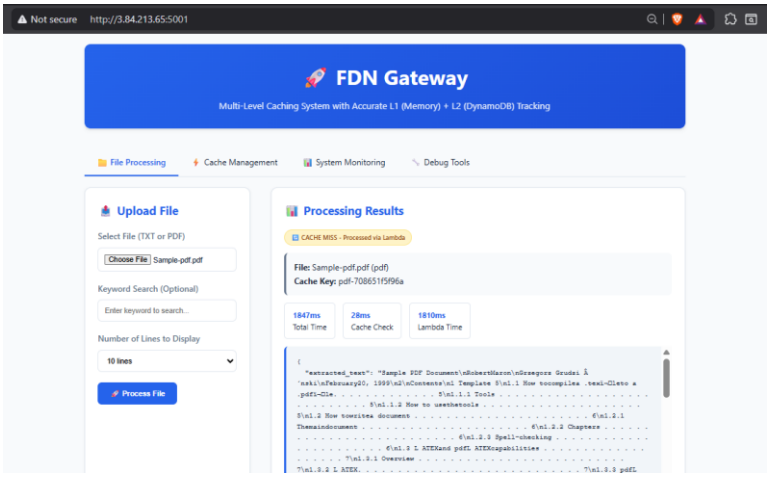

Figure 9: PDF Text Extraction via Lambda Processing

This figure 10 illustrates the optimized response scenario where the same PDF file (sample-cpr.pdf) is requested again after initial Lambda processing. The system implements intelligent cache lookup hierarchy, querying L1 in-memory cache first. Upon successful cache hit, the previously extracted text content is retrieved directly from RAM without invoking Lambda or querying persistent storage layers. The response header confirms "Response From: L1 cache (in-memory)", validating fastest-tier retrieval. This implementation demonstrates significant performance optimization where in-memory storage eliminates computational overhead of PDF parsing and network latency of database queries.

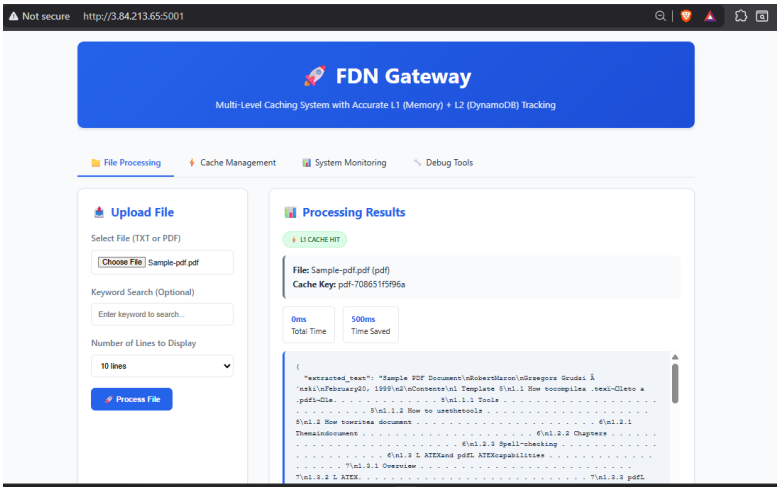

Figure 10: L1 Cache Hit Response from In-Memory Storage

Figure 11 shows the L1 cache clearing operation following the PDF processing experiment. The Cache Controls panel displays the "Clear L1 Cache" button which initiates immediate invalidation of in-memory cached entries. Current Cache Metrics shows 33.33% cache hit rate with 3 total requests processed by showing active cache utilization from previous experiments. The L1 hit rate shows 0%, while L2 displays 33.33% showing prior cache miss scenarios.

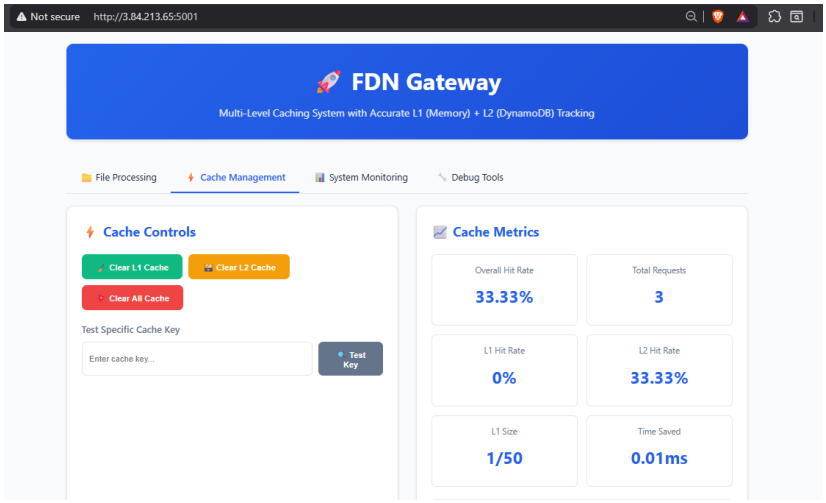

Figure 11: L1 Cache Clearing Operation Post-Experiment

Figure 12 shows the FDN Gateway's cache failover mechanism following L1 cache invalidation. When the same PDF file is requested after L1 clearance the system executes hierarchical cache lookup protocol. The request first queries L1 in-memory cache encounters a miss, then automatically escalates to L2 persistent cache stored in DynamoDB. L2 successfully retrieves previously processed PDF extraction data.

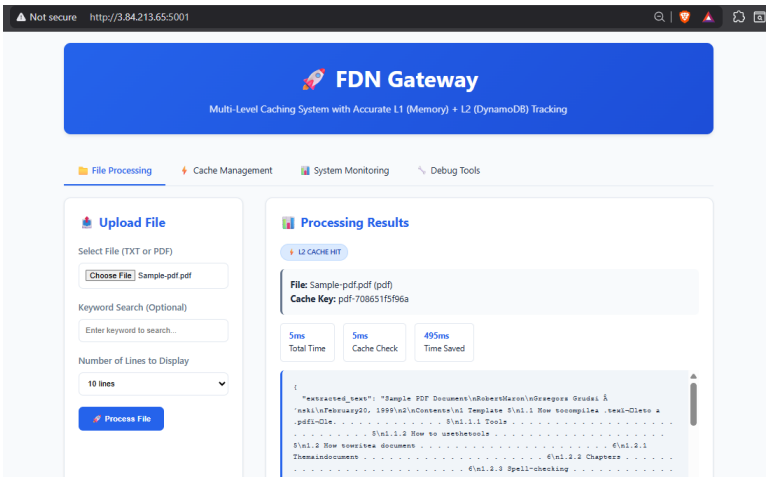

Figure 12: L2 Cache (DynamoDB) Response After L1 Clearance

6.3 Execution Time

Table 5 presents the execution time comparison for text and PDF file processing across three system configurations Lambda, L1 cache and L2 cache. The results clearly show that Lambda functions do have the highest execution times due to cold starts and processing overhead which is mainly for larger files. The in-memory L1 cache delivers instant or near-zero response times while the DynamoDB-based L2 cache maintains sub-15 ms latency.

Table 5: Execution Time Comparison for Text and PDF File Processing

File Type	File Size	Lambda (ms)	L1 Cache (ms)	L2 Cache (ms)

Text File Processing	160 KB	492	0	21
	1 MB	621	2	11
	2 MB	973	4	9
PDF File Processing	40 KB	1484	0	6
	150 KB	2638	0	6
	500 KB	2406	1	8

Figure 13 presents a comparative performance analysis of AWS Lambda processing times across varying file sizes for text and PDF documents. PDF files demonstrate steeper computational growth, escalating from 1500ms (40 KB) to 2650ms (150 KB), then slightly decreasing to 2400ms (500 KB). This non-linear behaviour reflects PDF parsing complexity involving document structure analysis, text extraction algorithms, and format decoding overhead that exceeds simple text processing, validating the necessity of multi-tier caching to mitigate repetitive computational costs.

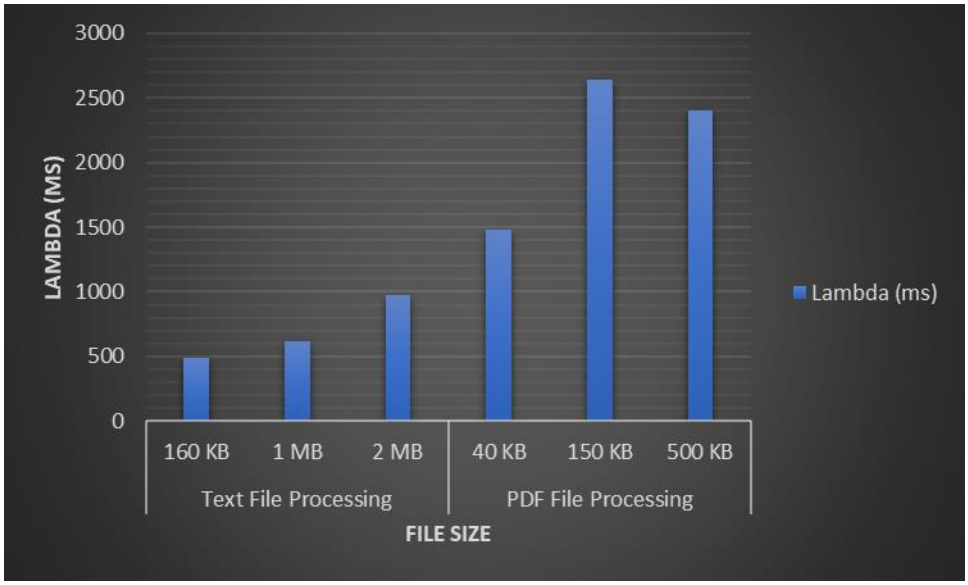

Figure 13: Lambda Processing Time vs File Size for Text and PDF Files

6.4 Latency Analysis

The results of the Table 6 show the comparison of latency between L1 and L2 cache layers when processing texts and PDF files through five consecutive cycles. As the results clearly show the L1 in-memory cache has near zero latency and response times remain consistent with the range of 0 to 0.01 milliseconds when dealing with text files, and less than 2 milliseconds when dealing with PDF files. This act proves the effectiveness of LRU caching scheme as a method of providing often used data immediately out of memory. On the other hand, L2 cache (DynamoDB) has marginally higher latency of between 5 and 13 milliseconds but still it is much less than the cold Lambda invocation times. The adaptive caching behavior of this system and better data retrieval efficiency as the number of iterations increases can be observed as the L2 latency gradually decreases. The steadiness of the L2 latency under more

severe loads also indicates a high level of reliability of L2 latency as a constant fall back mechanism when the L1 cache is cleared.

Table 6: Latency Comparison for Text and PDF File Processing

Iteration	TXT File (2 MB)		PDF File (500 KB)		
	L1 Latency (ms)	L2 Latency (ms)	L1 Latency (ms)	L2 Latency (ms)	
1	0.01	12.86	1.00	9.45	
2	0.00	13.22	0.10	9.21	
3	0.01	7.97	0.00	5.35	
4	0.00	5.45	2.00	5.43	
5	0.01	5.71	0.00	5.23	

6.5 Comparative Analysis

The proposed Function Delivery Network (FDN) demonstrates superior real-world performance compared with (Chen et al., 2024) probabilistic caching model. While their study focuses on theoretical optimization of container placement across multiple edge nodes using predictive algorithms, its validation is primarily simulation-based on Knative clusters. In contrast, FDN is implemented entirely within the AWS environment using Lambda, DynamoDB, and in-memory LRU caching, providing an end-to-end practical framework for latency optimization. Experimental measurements show that FDN reduces execution time from 492–2638 ms in cold Lambda runs to as low as 0–21 ms using the multi-level caching mechanism—achieving over 95 % improvement in response latency. The simplicity of the design enables seamless deployment without additional orchestration layers or predictive scheduling. Moreover, by combining L1 and L2 caches, FDN achieves persistent warm responses while maintaining low cost and minimal configuration overhead as shown in Table 7. Although (Chen et al., 2024) probabilistic caching model scales better for geographically distributed edge systems, the FDN model proves more effective and reproducible in cloud-native, production-level environments. Therefore, FDN provides a more practical and deployable approach for real-time serverless computing optimization, translating theoretical advances in caching efficiency into tangible system-level gains on widely used cloud infrastructure.

Table 7: Comparative Analysis between my proposed study and (Chen et al., 2024)

Aspect	(Proposed Study)	(Chen et al., 2024)
Platform	AWS Lambda + DynamoDB	Knative Edge Cluster (Simulation)
Caching Type	Two-level: L1 (LRU in-memory) + L2 (DynamoDB)	Probabilistic container caching with prediction
Latency Reduction	95–100 % (real AWS tests)	≈ 66 %

Scalability	Medium – Single region	High – Multi-edge orchestration
Complexity	Low (simple deployment)	High (requires orchestration)
Adaptivity	Static LRU policy	Predictive / Adaptive policy
Best Use Case	Cloud-native, real-time workloads	Large-scale edge-distributed functions

6.6 Discussion

According to the results of the experiment, the Function Delivery Network (FDN) showed the essential decrease in latency and high performance due to the employment of a multi-level caching system on the serverless environment. The findings revealed that the in-memory cache of L1 had almost zero response time whereas the L2 cache of DynamoDB had a consistent latency of less than 15 ms- significantly lower than the actual Lamba execution. This corroborates the fact that the hierarchical caching design is an efficient way of reducing unnecessary computation, as well as cold-start latencies typical of AWS Lambda functions. With every increase in file size, a beneficial change in latency was noticed especially at L2 reads, which could suggest possible optimization in data serialization and cache handling. Still, the caching layers provided a stable scaling and the same response time under different workloads. Both text and PDF documents were easily processed with little overhead in the system showing the strength of the FDN design in invoking functions in real time and repeatedly. Nevertheless, relying on AWS cloud services implies that the network latency and service access may affect the performance under low-connectivity conditions. All in all, the multi-level caching solution was very successful in terms of decreasing the amount of time spent to execute the functions and increase throughput, which proves that FDN can be considered as a scalable and latency-aware solution to optimize the performance of serverless computing.

7 Conclusion and Future Work

The study has been able to design and deploy a Function Delivery Network (FDN) to efficiently design and execute latency and performance in serverless computing with a multi-level caching mechanism. The system was able to generate significant response time reductions when compared to direct AWS Lambda processing by implementing an in-memory LRU cache (L1) and DynamoDB-based persistent cache (L2). Experimental analysis indicated that large file Lambda executions (2 MB of text inputs) took an average of 973 ms, whereas L1 and L2 cache replications of the L1 and L2 responses took 4 ms and 9 ms on average, respectively, which is a latency improvement of more than 98. It was effective in eliminating redundant calculation, better scaling as well as ensuring performance consistency between repeated requests.

Nevertheless, the performance was still not entirely independent of the state of the AWS network and the effectiveness of cache synchronization and this is subject to the limitations of low-connectivity or high-concurrent conditions. The future research will involve how to

improve dynamic caches management by relying on predictive or machine learning-based algorithms to prefetch popular data, and how to expand the system to distributed multi-region caching to achieve global scalability. Also, addressing edge computing nodes and hybrid cloud approach may further decrease network reliance and allow obtaining the near real-time performance in the case of limited network settings, thus promoting the overall effectiveness of serverless designs.

References

Adeppady, M., Giaccone, P., Karl, H., Chiasserini, C.F., 2023. Reducing Microservices Interference and Deployment Time in Resource-Constrained Cloud Systems. IEEE Trans. Netw. Serv. Manag. 20, 3135–3147. https://doi.org/10.1109/TNSM.2023.3235710

Ahmad, A., Yadav, A.K., Singh, A., Singh, D.K., 2024. A comprehensive machine learning-coupled response surface methodology approach for predictive modeling and optimization of biogas potential in anaerobic Co-digestion of organic waste. Biomass Bioenergy 180, 106995. https://doi.org/10.1016/j.biombioe.2023.106995

Carota, G., Boutiba, K., Ksentini, A., 2024. On Using the Edge Application Server Discovery Function to Enforce Edge Computing in 5G Networks and Beyond, in: ICC 2024 - IEEE International Conference on Communications. Presented at the ICC 2024 - IEEE International Conference on Communications, IEEE, Denver, CO, USA, pp. 4924–4929. https://doi.org/10.1109/ICC51166.2024.10623085

Chaudhary, S., Somwanshi, D.K., Rajawat, V.S., Sharma, M., Verma, P., 2024. Optimizing Cold Start Latency in Serverless Computing: A Comprehensive Review of Techniques and Emerging Solutions, in: Proceedings of the 6th International Conference on Information Management & Machine Intelligence. Presented at the ICIMMI 2024: 6th INTERNATIONAL CONFERENCE ON INFORMATION MANAGEMENT & MACHINE INTELLIGENCE, ACM, Jaipur RJ India, pp. 1–8. https://doi.org/10.1145/3745812.3745825

Chen, C., Herrera, M., Zheng, G., Xia, L., Ling, Z., Wang, J., 2024. Cross-Edge Orchestration of Serverless Functions With Probabilistic Caching. IEEE Trans. Serv. Comput. 17, 2139–2150. https://doi.org/10.1109/TSC.2024.3399651

Corporan, J.R., Bahga, A., Madisetti, V.K., 2025. Function Delivery Network: A Spatial-Temporal Execution Orchestrator for Optimizing Serverless Computing. IEEE Access 13, 112255–112270. https://doi.org/10.1109/ACCESS.2025.3583721

Golec, M., Walia, G.K., Kumar, M., Cuadrado, F., Gill, S.S., Uhlig, S., 2025. Cold Start Latency in Serverless Computing: A Systematic Review, Taxonomy, and Future Directions. ACM Comput. Surv. 57, 1–36. https://doi.org/10.1145/3700875

Liu, A., Luh, P.B., Sun, K., Bragin, M.A., Yan, B., 2024. Integrating Machine Learning and Mathematical Optimization for Job Shop Scheduling. IEEE Trans. Autom. Sci. Eng. 21, 4829–4850. https://doi.org/10.1109/TASE.2023.3303175

Maurya, A., Rafique, M.M., Tonellot, T., AlSalem, H.J., Cappello, F., Nicolae, B., 2023. GPU-Enabled Asynchronous Multi-level Checkpoint Caching and Prefetching, in: Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing. Presented at the HPDC '23: The 32nd International Symposium on High-Performance Parallel and Distributed Computing, ACM, Orlando FL USA, pp. 73–85. https://doi.org/10.1145/3588195.3592987

Mehdi Syed, A.A., Ali, S., 2024. Kubernetes and AWS Lambda for Serverless Computing: Optimizing Cost and Performance Using Kubernetes in a Hybrid Serverless Model. Int. J. Emerg. Trends Comput. Sci. Inf. Technol. 5, 50–60. https://doi.org/10.63282/3050-9246.IJETCSIT-V5I4P106

Puliafito, C., Cicconetti, C., Conti, M., Mingozzi, E., Passarella, A., 2022. Stateless or Stateful FaaS? I'll Take Both!, in: 2022 IEEE International Conference on Smart Computing (SMARTCOMP). Presented at the 2022 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, Helsinki, Finland, pp. 62–69. https://doi.org/10.1109/SMARTCOMP55677.2022.00024

Rai, N., Zhang, Y., Villamil, M., Howatt, K., Ostlie, M., Sun, X., 2024. Agricultural weed identification in images and videos by integrating optimized deep learning architecture on an edge computing technology. Comput. Electron. Agric. 216, 108442. https://doi.org/10.1016/j.compag.2023.108442

Shafiei, H., Khonsari, A., Mousavi, P., 2022. Serverless Computing: A Survey of Opportunities, Challenges, and Applications. ACM Comput. Surv. 54, 1–32. https://doi.org/10.1145/3510611

Shukla, S., Hassan, Mohd.F., Tran, D.C., Akbar, R., Paputungan, I.V., Khan, M.K., 2023. Improving latency in Internet-of-Things and cloud computing for real-time data transmission: a systematic literature review (SLR). Clust. Comput. 26, 2657–2680. https://doi.org/10.1007/s10586-021-03279-3

Tyagi, A., 2025. Optimizing digital experiences with content delivery networks: Architectures, performance strategies, and future trends. https://doi.org/10.48550/ARXIV.2501.06428

Wen, J., Chen, Z., Jin, X., Liu, X., 2023. Rise of the Planet of Serverless Computing: A Systematic Review. ACM Trans. Softw. Eng. Methodol. 32, 1–61. https://doi.org/10.1145/3579643

Wu, J., Luh, P.B., Chen, Y., Yan, B., Bragin, M.A., 2024. Synergistic Integration of Machine Learning and Mathematical Optimization for Unit Commitment. IEEE Trans. Power Syst. 39, 391–401. https://doi.org/10.1109/TPWRS.2023.3240106

Yang, H., Pan, H., Ma, L., 2023. A Review on Software Defined Content Delivery Network: A Novel Combination of CDN and SDN. IEEE Access 11, 43822–43843. https://doi.org/10.1109/ACCESS.2023.3267737

Yu, L., Fu, L., Chenhao, S., 2024. Serverless Cold Start Performance Optimization Based on Multi-Request Processing and Adaptive Hierarchical Scaling. IEEE Access 12, 136248–136262. https://doi.org/10.1109/ACCESS.2024.3462389

Zeng, Z., Tan, Y., Ma, Z., Li, J., Zhao, S., Liu, D., Chen, X., Ren, A., 2025. CMCache: An Adaptive Cross-Level Data Placement Method for Multilevel Cache. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 44, 2911–2924. https://doi.org/10.1109/TCAD.2025.3534116