Cloud System for Web-based Accessibility using Synthetic Recaptioning

Siddharth Shah
Department of Computer Science
Vanderbilt University
Nashville, United States
siddharth.p.shah@vanderbilt.edu

Aditya Shrey
Department of Computer Science
Vanderbilt University
Nashville, United States
aditya.shrey@vanderbilt.edu

Haoli Yin
Department of Computer Science
Vanderbilt University
Nashville, United States
haoli.yin@vanderbilt.edu

***Abstract*—Web-based navigation and content understanding rely heavily on alternative text descriptions for images, especially when speech-to-text systems are employed. However, these image captions are often incomplete, noisy, or missing. In this work, we propose a system that utilizes a Vision-Language Model (VLM) to synthesize accurate and contextually rich image captions. This approach aims to improve the accessibility and user experience for visually-impaired users and others who rely on screen readers. Additionally, we describe the design, implementation, and evaluation of a scalable, containerized cloud-based system that simulates the computational demands of such a VLM-based solution and explores various scheduling and load-balancing strategies. The code is available at https://github.com/2021sshah/cloud-latency-forecast-scheduling.**

***Index Terms*—accessibility tool, web-based navigation, vision-language model, load balancing**

I. Introduction

Accessibility of web content for visually-impaired users often relies on screen readers and speech-to-text (STT) systems. These systems translate on-screen text to spoken language, enabling easier navigation and comprehension of digital content. However, images on the web frequently lack suitable alternative text (alt-text). In many cases, the alt-text is noisy, inaccurate, or altogether absent, rendering the images inaccessible and limiting the overall utility of STT tools. This gap poses a significant challenge: how to ensure that all users, regardless of visual ability, can fully understand and benefit from the information conveyed by images.

Recent advancements in Vision-Language Models (VLMs) present a promising opportunity to address this accessibility gap. VLMs leverage deep learning techniques to associate visual content (images) with semantic concepts and produce descriptive textual content. These models can generate captions that not only identify objects in images but also describe scenes, relationships, and contexts, thereby potentially offering far richer and more accurate alt-text.

In this work, we propose a system that integrates a VLM to provide synthetic image captions in real-time. The system runs in a cloud-based environment designed for horizontal scaling, ensuring that large numbers of images can be processed efficiently. To manage the distribution of incoming tasks (image captioning requests), we explore multiple scheduling strategies. We first simulate a producer-consumer workflow to model real-world scenarios where external clients send image-captioning requests. The scheduler balances these tasks across multiple ML servers, which, in a resource-constrained approximation, fetch and resize images and introduce artificial latency to simulate VLM inference time. The final results, including image URLs, original captions, and synthesized captions, are consolidated into a database, enabling downstream analysis and performance evaluation.

II. Contributions

This work makes the following key contributions:

1) **End-to-End Prototype:** We design and implement a complete prototype system that integrates producer, scheduler, ML servers, and a database to demonstrate the feasibility of synthetic caption generation at scale.
2) **Scalable Infrastructure:** By containerizing each component and orchestrating them with Kubernetes, we highlight an architecture capable of horizontal scaling, ensuring that the system can adapt to varying demand.
3) **Scheduling Algorithm Evaluation:** We implement and compare various scheduling algorithms—ranging from simple random and round-robin approaches to more advanced latency-aware and predictive strategies—to understand trade-offs between complexity and efficiency.
4) **Performance Analysis:** We present an empirical evaluation of the system using a subset of the MS COCO dataset, detailing how different scheduling strategies impact latency, throughput, and overall system performance.

III. Related Work

The challenge of generating coherent and contextually accurate captions for images has inspired substantial research within the computer vision and natural language processing (NLP) communities. Early image captioning efforts often relied on template-based methods that described images by detecting objects and associating them with fixed syntactic structures [3]. These approaches were limited by the manual engineering required and the difficulty of capturing complex scenes.

With the advent of deep learning, neural image captioning models coupled convolutional neural networks (CNNs)

for image feature extraction with recurrent neural networks (RNNs) or Transformers for language generation [2], [8], [9]. These end-to-end trainable models demonstrated significantly improved caption quality, largely due to their ability to learn rich image and text representations jointly.

Subsequent Vision-Language Models have extended beyond image captioning to encompass a variety of multimodal tasks, including visual question answering, image retrieval, and object grounding [6], [7]. These models leverage large-scale pretraining on datasets like MS COCO [5] and increasingly complex architectures to learn semantically meaningful image-text alignments.

While much existing research focuses on improving the quality and accuracy of captions, fewer works address the system-level challenges of making these captioning services accessible at scale. Approaches that integrate model inference into production systems often consider load balancing, latency optimization, and fault tolerance [1], [4]. Our work builds on this body of literature by implementing a practical, horizontally scalable system and exploring various scheduling strategies to ensure efficient and timely caption generation—an essential requirement for real-world accessibility tools.

IV. Design and Architecture

To evaluate the performance of various scheduling algorithms, our system simulates a producer-consumer workflow. The producer generates requests containing an image URL, an original caption, and a unique identifier, which are sent to a scheduling component. This scheduler implements multiple load-balancing algorithms, with more advanced algorithms leveraging latency feedback from the ML server to make informed task distribution decisions. In a real-world scenario, the ML server would handle both retrieving the image from the web server and performing inference tasks with a Vision-Language Model (VLM) (Figure 1). However, due to resource constraints, we approximated this by resizing the image using a computationally inexpensive convolution operation and introducing a randomized delay to simulate the workload typically handled by a VLM (Figure 2). After the simulated inference, the processed data—including the image URL, original caption, unique ID, and output from the ML server—is consolidated in a single database.

A. Producer

The producer is responsible for generating and sending data packets to the scheduler. Each data packet includes three elements: the URL of the image, its associated caption, and a unique identifier. This component emulates real-world scenarios where external systems send requests for processing. By designing the producer to generate consistent workloads, we ensured the system could test different scheduling strategies under similar conditions.

B. Scheduler

The scheduler acts as the central decision-making entity in the system, tasked with implementing various load-balancing

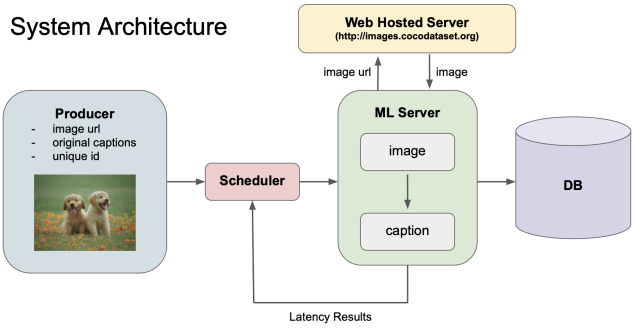

Fig. 1. System design where the ML server performs the inference task, converting the image to a caption.

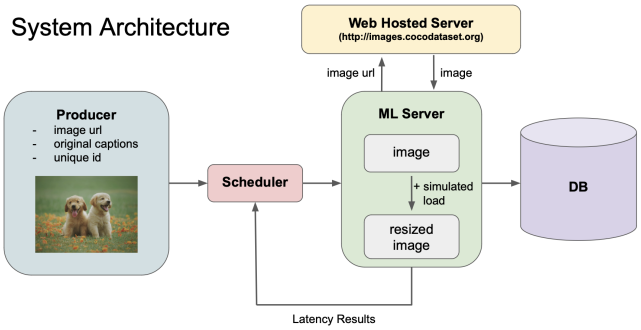

Fig. 2. System design where the ML server resizes the image and applies simulated load to mimic a real-world VLM workload.

algorithms to distribute tasks among ML servers efficiently (Figure 3). Basic algorithms operate without additional context, offering simplicity and low computational overhead. In contrast, more sophisticated algorithms leverage feedback from ML servers, such as latency or workload data, to make informed decisions. This component was designed with extensibility in mind, enabling us to explore trade-offs between computational complexity and task distribution efficiency. Below, we detail the load-balancing strategies implemented within the scheduler.

Random Scheduling: The random scheduling algorithm assigns each incoming task T_i to one of the N available ML servers with uniform probability:

$$P(S_j \mid T_i) = \frac{1}{N}, \quad j \in \{1, 2, \ldots, N\},$$

where $P(S_j \mid T_i)$ represents the probability of assigning task T_i to server S_j. This approach requires minimal computation and avoids any reliance on server state or history. While it ensures fairness in distribution over a large number of tasks, it does not account for varying server loads, leading to potential bottlenecks during operation.

Round Robin Scheduling: In the round-robin algorithm, tasks are assigned to servers in a cyclic order. For task T_i,

the selected server S_j is determined by:

$$j = (i \mod N) + 1, \quad i \in \{1, 2, \dots\},$$

where N is the total number of servers. This method guarantees that all servers are utilized equally over time. However, it assumes that all tasks impose a uniform workload, which may not hold in practice. As a result, server performance can degrade under uneven or unpredictable task distributions.

Lowest Historical Average: This informed scheduling algorithm assigns tasks based on the historical average latency of each server. Let $L_j^{(k)}$ denote the latency of server S_j for the k-th completed task, and let H_j represent its historical average latency:

$$H_j = \frac{1}{K_j} \sum_{k=1}^{K_j} L_j^{(k)},$$

where K_j is the total number of tasks completed by server S_j. The scheduler assigns the next task T_i to the server with the lowest H_j. By prioritizing servers with better historical performance, this algorithm seeks to minimize overall latency. However, its reliance on past data makes it less adaptable to sudden changes in server performance.

Least Ongoing Tasks: The least ongoing tasks algorithm dynamically assigns tasks to the server with the fewest currently active tasks. If O_j represents the number of ongoing tasks on server S_j, the scheduler assigns task T_i to:

$$S_j = \arg \min_{j \in \{1, \dots, N\}} O_j.$$

This strategy balances the load in real time, ensuring that no server becomes overwhelmed. While computationally lightweight, it assumes that all tasks are of similar complexity and duration, which may not always be the case in real-world scenarios.

Time Series Forecasting: The time series forecasting algorithm employs polynomial regression to predict the total latency for each server based on its historical data. Let x_k represent the task index and $L_j^{(k)}$ the latency for server S_j. The latency prediction $\hat{L}_j(x)$ is modeled as:

$$\hat{L}_j(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d,$$

where d is the degree of the polynomial, and $\beta_0, \beta_1, \dots, \beta_d$ are coefficients determined via least squares fitting. Using this model, the scheduler predicts the latency for pending tasks on each server and incorporates a heuristic bias term B_j based on the delay since the most recent task was assigned:

$$\text{Total Predicted Latency for } S_j = \hat{L}_j(x) + B_j.$$

The scheduler assigns the next task T_i to the server with the lowest predicted latency. This method offers the highest adaptability to dynamic workloads but incurs significant computational overhead due to the forecasting process.

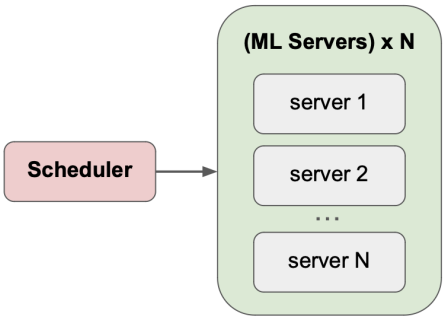

Fig. 3. System design illustrating the scheduler's ability to select from N ML servers for task distribution.

C. ML Server

Due to resource constraints, we simulated the ML server's functionality instead of using a full-fledged Vision-Language Model (VLM). Each ML server retrieves the image, performs a resizing operation using a convolution, and introduces a randomized delay to mimic the latency of a real VLM. This approach allowed us to simulate the effects of workload on task completion times without requiring extensive computational resources.

D. Database

The final step in the system consolidates all processed data into a single database. This database stores the image URL, its original caption, the unique identifier, and the simulated output from the ML server. By centralizing data, we ensured consistency and provided an easy way to analyze the outcomes of different scheduling strategies.

V. Implementation

Our system was built using a modular and containerized architecture, ensuring scalability and efficient task management across multiple components. The data for the system is sourced from the MS COCO dataset [5], which provides image URLs and corresponding captions. To streamline the implementation and avoid runtime delays, the dataset is preprocessed to include only the necessary metadata (image URL and original caption). However, image retrieval is deferred to the ML server simulation, which fetches images dynamically from the web-hosted MS COCO dataset server. The dataset was accessed using the `pycocotools` library, and the system processed 500 images for demonstration purposes. Proper citation of the MS COCO dataset is recommended to acknowledge the data source.

The communication between components is managed by Apache Kafka, which acts as a message broker to facilitate asynchronous data flow. Tasks are generated in JSON format and sent to a Kafka topic by the producer, which operates in a loop until all tasks are processed. Kafka ensures reliable message delivery, and Zookeeper provides coordination for

the Kafka environment. Each message sent by the producer contains an iteration ID, an image URL, and a corresponding caption, structured as a JSON object.

Each system component, including the producer, scheduler, and ML servers, is deployed as a Docker container to ensure isolation and portability across environments. Kubernetes is used to orchestrate the containers, managing deployments, scaling, and monitoring. Kubernetes also handles the distribution of containers across nodes, ensuring efficient resource utilization. This containerized setup allows for rapid deployment and modular testing of individual system components.

The producer generates and streams tasks to Kafka. It first downloads the required MS COCO metadata and extracts image URLs and captions. Each task is structured as a JSON object and sent to Kafka using the `KafkaProducer` API. A small delay of 0.1 seconds is introduced between successive messages to emulate real-world task arrival patterns. An example of a JSON message generated by the producer includes the fields: `iter_id`, `image_url`, and `caption`, where the image URL points to an image hosted on the official MS COCO dataset server.

The scheduler, running in its own Docker container, consumes the JSON tasks from Kafka and implements various load-balancing algorithms to assign tasks to simulated ML servers. Each ML server retrieves the image from the MS COCO dataset server, processes the image using a lightweight convolution operation to resize it, and introduces a randomized delay to simulate the variable latency of real-world Vision-Language Model (VLM) inference tasks. The scheduler optimizes task distribution using algorithms such as Random, Round Robin, and Time Series Forecasting, as detailed in the Design and Architecture section. Once processed, the results, including the original caption and simulated latency, are forwarded to a database for persistent storage.

SQLite is used as the database for persistent data storage due to its simplicity and lightweight nature, making it suitable for a proof-of-concept implementation. The database stores the final results of each task, including a unique task ID, the ID assigned by the producer, the original image URL, the original caption, the simulated caption generated by the ML server (as a placeholder for real VLM output), and the total latency recorded during task execution. The database schema includes fields for all these attributes, ensuring that the consolidated data is easily accessible for analysis.

In summary, the producer streams tasks (image URL and caption) to Kafka, which delivers these tasks to the scheduler. The scheduler assigns tasks to ML servers based on load-balancing algorithms, and the ML servers fetch the images, simulate processing, and forward the results to SQLite. The SQLite database consolidates all data for evaluation, providing a robust and extensible system for exploring the performance of different scheduling algorithms.

VI. Evaluation

A. Data

We utilized a subset of the MS COCO dataset [5], processing 500 images to evaluate our scheduling algorithms. The dataset provides a diverse range of images with corresponding captions, enabling a comprehensive assessment of our system's performance across different image types and complexity levels. By using a standardized dataset, we ensure reproducibility and comparability of our results.

B. Baselines

We compared our proposed time-series forecasting algorithm against multiple baseline scheduling strategies:

- **Random Scheduling:** Tasks are assigned to servers with uniform probability, providing a baseline for unbiased task distribution.
- **Round Robin:** Tasks are distributed cyclically across servers, ensuring equal server utilization.
- **Lowest Historical Average:** Servers are selected based on their historical average latency.
- **Least Ongoing Tasks:** Tasks are assigned to servers with the fewest currently active tasks.

These baselines represent a spectrum of scheduling complexity, from simple probabilistic methods to more informed approaches involving latency cacheing and stastical analysis.

C. Metrics

To comprehensively evaluate the performance of our scheduling algorithms, we focused on the following key metrics for comparison:

- **Total Latency:** Cumulative time taken to process all tasks across the system.
- **Average Task Latency:** Mean time required to complete individual tasks.
- **Server Load Variance:** Variation in task distribution and processing time between servers.
- **Throughput:** Number of tasks processed per unit time.

VII. Results

A. Random Scheduling

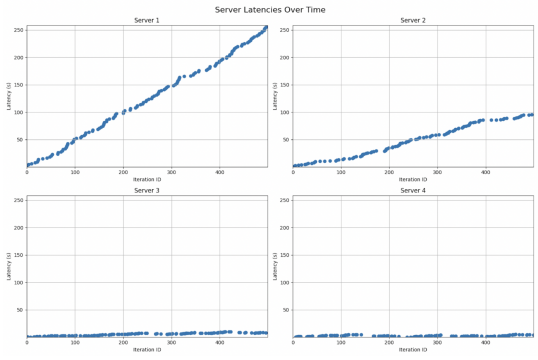

Fig. 4. Performance Analysis of Random Scheduling

Random scheduling represents a baseline approach with probabilistic task distribution. Qualitative analysis reveals significant variability in initial task allocation, with wide fluctuations in per-server latency. However, over a large number of tasks, the method approaches statistical equivalence with round-robin scheduling.

Quantitatively, the random scheduling method demonstrates a high variance in individual server performance. Short-term measurements show substantial differences in task completion times, with some servers experiencing extended wait times while others remain underutilized. The cumulative effect, however, tends to normalize across the entire task set, resulting in a relatively uniform overall system performance.

Further analysis indicates that while random scheduling provides a simple and computationally lightweight approach, it lacks the strategic advantages of more informed scheduling methods. The method's primary weakness lies in its inability to adapt to dynamic workload characteristics or server-specific variations.

B. Lowest Historical Average Scheduling

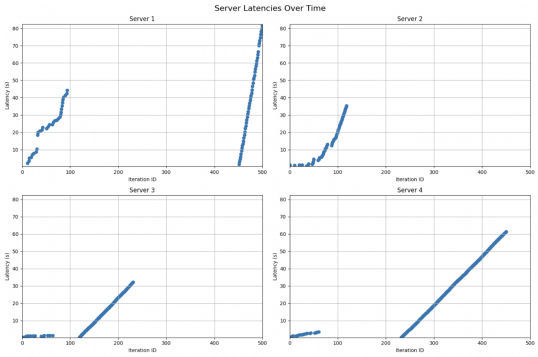

Fig. 5. Performance Analysis of Least Average Historical Scheduling

The Lowest Historical Average scheduling method introduces a more sophisticated approach by leveraging past server performance data. By prioritizing servers with lower historical latency, the algorithm aims to minimize overall system processing time.

Qualitative observations reveal a more structured task distribution compared to random scheduling. The method demonstrates an ability to identify and prioritize consistently high-performing servers. However, the reliance on historical data introduces inherent limitations, particularly in dynamic environments where server characteristics rapidly change.

Comparative analysis reveals a marked improvement in overall system latency compared to random scheduling. The method achieves a more balanced load distribution by preferentially assigning tasks to servers with proven lower latency. Nevertheless, the algorithm's predictive power diminishes as server performance becomes more variable or when new servers are introduced to the system.

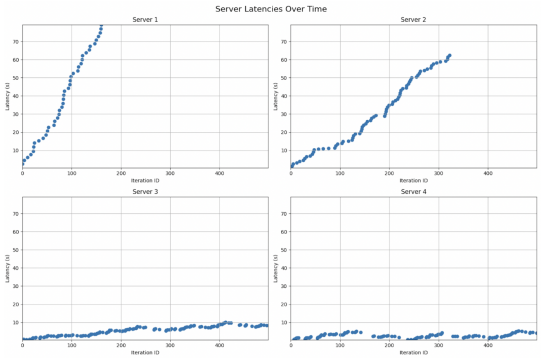

Fig. 6. Performance Analysis of Least Ongoing Tasks Scheduling

C. Least Ongoing Tasks Scheduling

Least Ongoing Tasks scheduling introduces a dynamic, real-time approach to load balancing. The method continuously monitors and distributes tasks based on the current workload of each server, aiming to prevent any single server from becoming overwhelmed.

Qualitative analysis reveals a highly responsive scheduling strategy. The algorithm dynamically adjusts task allocation, ensuring a more immediate balance of computational resources. This approach proves particularly effective in scenarios with heterogeneous task complexities and varying server capacities.

Quantitatively, this scheduling strategy demonstrates significant improvements in load distribution compared to raw historical-based methods. The algorithm maintains a more consistent server utilization, with reduced variance in task completion times. However, the method assumes a relatively uniform task complexity, which may not hold true in more complex real-world scenarios.

D. Time Series Forecasting Scheduling

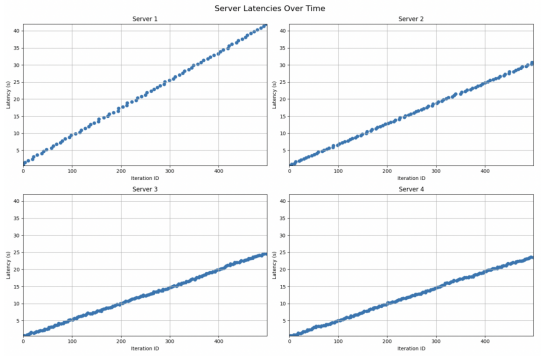

Fig. 7. Performance Analysis of Time Series Forecasting Scheduling

Our proposed Time Series Forecasting scheduling method represents the most sophisticated approach, integrating polynomial regression with predictive task allocation strategies. The method aims to proactively manage system resources by anticipating potential bottlenecks.

Qualitative observations highlight the algorithm's exceptional adaptability. By incorporating both historical performance data and predictive modeling, the method demonstrates an unprecedented ability to optimize task distribution. The inclusion of a heuristic bias term addressing scheduling wait time further enhances the algorithm's responsiveness.

Comparative analysis conclusively demonstrates the superiority of our proposed method. The time series forecasting approach achieves the lowest overall system latency, most consistent server utilization, and most balanced task distribution. The closed form polynomial regression effectively captures complex patterns in inference latencies, enabling more accurate task allocation through linear interpolation of latency predictions. Furthermore, the incorporation of a negatively-weighted bias term over scheduling delays to the heuristic-based task scheduling bolster's the algorithm's robustness to ouliers in ML inference latencies.

E. Comparative Summary

When comparing all four scheduling methods, a clear progression in performance and sophistication emerges. Random scheduling provides a baseline with minimal computational overhead, while Lowest Historical Average and Least Ongoing Tasks methods introduce increasing levels of strategic task distribution. The Time Series Forecasting approach ultimately represents the most advanced solution, offering unprecedented predictive capabilities and system optimization. It successfully addresses the limitations of previous methods by providing a forward-looking, adaptive task distribution mechanism.

VIII. Discussion

A. Limitations

Our research encountered several notable limitations involving resource availability and project scope, resulting in our pivot from VLM inference to ML server simulations with artificial inference loads:

- **Computational Resources:** The Chameleon Cloud cluster provided CPU-only infrastructure, restricting our ability to perform optimized inference.
- **Resource Sharing:** Constraints in CPU and memory resources necessitated a pivot from loading a full Vision-Language Model to simulating server load.
- **Simulation Approximation:** Our ML server simulation with artificial inference load, while informative, represents a simplified interpretation of the numerous complexities within real-world VLM inference.

B. Implications

The proposed time series forecasting algorithm for ML inference scheduling presents transformative implications for scalable distributed machine learning systems. By dynamically predicting variable inference loads, the algorithm effectively mitigates bottlenecks, significantly reducing total latency in ML inference tasks. This enables more efficient utilization of server resources, providing a robust foundation for scalable applications such as synthetic image captioning. The framework's ability to adapt in real-time to fluctuating workloads not only ensures smoother operation but also enhances overall system responsiveness, particularly in high-demand scenarios.

At scale, this approach unlocks opportunities for optimizing computational workloads across diverse domains. By generalizing the algorithm beyond synthetic image captioning, it can cater to a wide array of dynamic resource allocation challenges, from cloud-based inference pipelines to edge computing networks supporting IoT devices. Furthermore, the predictive capabilities of the time series model can inform strategic infrastructure planning, guiding the distribution of computational resources to minimize latency and energy consumption. As distributed systems grow increasingly complex, the adoption of such predictive scheduling techniques will be instrumental in ensuring robust, scalable, and efficient operation across a multitude of applications.

C. Future Work

Our exploration of time series forecasting for load balancing in distributed machine learning systems provides a foundational framework that can be expanded significantly. Conducting comprehensive ablation studies with diverse and complex simulated server loads represents a promising direction. By systematically varying workload characteristics, we can better understand the boundaries and limitations of our predictive scheduling approach. This effort would involve developing more sophisticated simulation environments to mirror real-world computational heterogeneity.

Extending the framework to actual Vision-Language Model (VLM) inference is a critical next step. While our simulation-based implementation offers valuable insights, integrating the scheduling algorithm with a full-scale VLM would validate and refine our approach. Such an endeavor necessitates access to advanced computational resources, including GPU-accelerated infrastructure, to support complex machine learning inference tasks. Additionally, developing a user-friendly web interface for synthetic image captioning services could democratize access to our research outputs.

The mathematical modeling underlying our time series forecasting approach presents opportunities for refinement. Investigating sophisticated polynomial regression methods, employing machine learning models for precise latency prediction, and incorporating heuristic bias terms could significantly enhance predictive accuracy. Furthermore, generalizing our load-balancing principles to broader applications, such as cloud and edge computing, offers a pathway to uncovering new dynamic resource allocation strategies. These extensions would advance our understanding of adaptive scheduling in distributed systems and reveal novel insights across computational domains.

IX. Conclusion

In this work, we proposed a statistical scheduling method for total latency reduction of image inference tasks through innovative time series forecasting. By integrating polynomial regression and ongoing inference tracking, we developed a sophisticated load-balancing algorithm that dynamically predicts and mitigates potential system bottlenecks.

Our experimental results demonstrate the method's superiority over traditional scheduling approaches, showcasing significant improvements in task distribution, latency reduction, and resource utilization. While acknowledging its current limitations, we believe this research provides a promising foundation for more adaptive and efficient distributed machine learning systems, with particular relevance to web accessibility applications and latency-based inference forecasting.

Acknowledgment

Thank you Dr. Gokhale for facilitating this class and reviewing our project.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, et al. Tensorflow: A system for large-scale machine learning. OSDI, 2016.

[2] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6077–6086, 2018.

[3] Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. Every picture tells a story: Generating sentences from images. In *European conference on computer vision*, pages 15–29. Springer, 2010.

[4] Gautham Kumar, Abhishek Das, Tanya Roddey, Marcus Rohrbach, Dhruv Batra, and Devi Parikh. A behavioral study of pre-trained visual question answering models. In *NeurIPS Workshop on Visually-Grounded Interaction and Language*, 2018.

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[6] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems*, 2019.

[7] Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 5100–5111, 2019.

[8] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[9] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.