Intelligent Model Switching System for Cloud and Edge Integration

Krishak Aneja 2023101106
Saiyam Jain 2023101135
Varun Gupta 2023101108
Gracy Garg 2023101118

I. Introduction

The goal of this project is to create a system that intelligently selects between models running on cloud and those running on the edge during runtime, dynamically optimizing performance while ensuring accurate, efficient, and environmentally sustainable results.

II. Problem Statement

Question Answering (QA) is a challenging Natural Language Processing (NLP) task, with models requiring substantial computational resources for processing. The ability to select the best-suited model—whether on the edge or in the cloud—based on performance and environmental considerations is critical. As technology evolves and more powerful hardware becomes available, it is essential to ensure that the system remains both efficient and adaptable.

III. Approach

The project uses a switching logic that takes inspiration from the MLFQ scheduling policy. This policy classifies models into different priority levels, dynamically selecting a model based on the query type and other relevant factors. By balancing model efficiency with environmental sustainability, the system reduces unnecessary resource consumption and ensures that the application remains effective as hardware improves.

We utilize four distinct Question Answering models: two local (edge-based) and two cloud-based models. These models include BERT, Gemini, Ollama, and the Qualcomm-provided Q&A model [1].

We calculate a "Worth Score" by assessing factors such as query characteristics, performance metrics, and environmental impact. The weight given to each metric is determined by the system's current state, enabling context-sensitive decision-making. This score guides the selection of the most appropriate model for each task. To refine the decision-making process, we use an exponential moving weighted average(EMWA) score, incorporating the new score into previous scores . This approach, inspired by reinforcement learning, enables the system to adapt and improve model selection over time, ensuring optimal performance.

IV. Model-Switching Logic and Optimization

The system incorporates a robust switching logic, ensuring that the application adapts to changes in hardware availability and performance improvements. The switching logic dynamically chooses the most appropriate model for each query, based on:

- Model priority: Models are prioritized based on their suitability for the query type based on factors like CPU usage, temperature, battery level and time taken to execute.
- Previous performance: Models that have performed well in the past are favored. Models that are rejected by the user repeatedly are penalized.
- Client feedback: User preferences and feedback help determine the ideal model for subsequent queries.
- Environmental impact: The switching logic considers the environmental impact of running a model, ensuring sustainability

The system has been further optimized by embedding a caching mechanism directly within some models. This model-level caching enables efficient storage and retrieval of frequently accessed inputs and their corresponding outputs, thereby minimizing redundant computations. By leveraging this approach, the system effectively handles repeated or identical queries, significantly reducing processing overhead and latency. This design not only improves runtime efficiency but also enhances the system's responsiveness, particularly in scenarios involving high-frequency inputs or repeated sample queries.

V. Score Calculation Methodology

The `calculateScore` method implements a weighted scoring system to evaluate model performance under various environmental and operational conditions. This section describes the method's functionality, breaking down its components and their contributions to the final score.

A. Overview

The method computes a score based on:

- **Battery level**: Indicates the device's remaining power.
- **Battery consumption**: Represents energy usage during processing.
- **Temperature**: Reflects thermal conditions.
- **CPU usage**: Assesses processing load.
- **Token factor**: Denotes model affinity for input size.
- **Feedback**: Captures user satisfaction levels.

B. Weighted Contributions

Weights for battery and CPU usage adapt dynamically:

- A battery level below 30% assigns a higher weight (0.4) to energy efficiency; otherwise, 0.2.
- If the temperature exceeds 40°C, CPU efficiency is emphasized with a weight of 0.4; otherwise, 0.2.
- Remaining weight is allocated to user feedback, ensuring its proportional influence.

C. Normalization and Score Calculation

The method normalizes scores for battery consumption and CPU usage, ensuring all metrics fall within a comparable range:

$$batteryScore = 1.0 - \frac{\text{batteryConsumption}}{100.0} \tag{1}$$

$$cpuScore = 1.0 - \frac{\text{cpuUsage}}{100.0} \tag{2}$$

The final score is calculated as:

$$\begin{aligned} baseScore =& (batteryScore \cdot \text{weightBattery}) \\ &+ (cpuScore \cdot \text{weightCpuUsage}) \\ &+ (feedback \cdot \text{weightFeedback}) \end{aligned} \tag{3}$$

$$finalScore = \text{eFactor} \cdot \text{tokenFactor} \cdot baseScore \tag{4}$$

The product of `eFactor` and `tokenFactor` scales the score to reflect environmental efficiency and model compatibility.

This method ensures a dynamic and adaptive scoring system, crucial for optimizing model selection under varying environmental and user-centric constraints.

VI. Application Framework

We have developed an Android application for On-Device Question Answering as the base framework of the project. The app provides two modes for model selection:

- Adapt Mode: This mode allows the app to automatically select the best model based on real-time performance and user feedback.
- Manual Mode(Local/Cloud): This mode allows the user to decide to only use a particular model for answer generation whilst still updating the EMWA score for future adaptive querying.

VII. Analysis

To validate the effectiveness and correctness of the system, we tested the application with various datasets and scenarios with the results and parameters of testing forwarded to a server for graphing. The graphs and other analyses can be accessed at the following repository: Graphs Repository.

VIII. Results and Evaluation

The application provides the flexibility to handle both small and large queries, optimizing between edge and cloud models to achieve the best possible response time and accuracy. By leveraging the cache, the system ensures that switching between models is efficient, reducing latency and improving the overall user experience.

IX. Conclusion

This project successfully demonstrates a dynamic, environmentally sustainable, and efficient approach to model selection between edge and cloud resources. The incorporation of an intelligent MLFQ-inspired policy as well as a caching mechanism facilitates the answering of the user's queries with maximum accuracy and minimized power overhead. Our approach is tolerant of changes in the model set, ensuring that the system can scale and adapt as hardware improves and new models become available, offering a solution that remains relevant in an evolving technological landscape.

Individual Contributions

- **Krishak Aneja**: Designed and implemented the switching logic, MLFQ scheduling and scoring metric. Also developed the user interface.
- **Saiyam Jain**: Handled the edge model's lifecycle, including loading, interpreting, input tokenization, prediction as well as query caching.
- **Varun Gupta**: Handled cloud-based functionalities, including loading the cloud model, integrating cloud API calls within the app, and managing local model loading and usage on the server PC.
- **Gracy Garg**: Responsible for graphing and the preparation of analyses and reports.

Acknowledgment

We would like to express our sincere gratitude to Professor Karthik Vaidhyanathan for providing us with the opportunity to work on this project and for his continuous guidance and support. We also extend our heartfelt thanks to our teaching assistant Ayush Raghuvanshi and our kind senior Arya Marda for their invaluable assistance throughout the project. We sincerely thank Qualcomm for their generous support and valuable resources that contributed to the success of this project. Working with the QIDK provided us with a valuable learning experience and enhanced our understanding of practical applications in this domain.

References

[1] The core model of this project is the Electra Transformer model fine-tuned on the SQUAD v2.0 dataset, which is a small, efficient, and mobile-friendly model designed for Question Answering tasks. This model has been accelerated using the Qualcomm Neural Processing SDK for AI framework, providing the necessary tools for on-device model inference.