Optimization of Inference Time for Large-Scale Language Models

Andrii Korniienko - 24103053
University of Bern, Switzerland
andrii.korniienko@students.unibe.ch

Abstract

In this paper, I focus on optimizing the inference time of large language models (LLMs), specifically GPT-2 variants, using caching techniques. I implement adaptive caching strategies to reduce memory overhead and inference latency during online serving of language models. The impact of caching, batch size, input length, and model configuration is explored through a comprehensive Design of Experiments (DoE) analysis. Using ANOVA and regression modeling, I identify the most significant factors affecting performance. Additionally, I simulate an online multi-user queueing scenario (M/M/K) to evaluate the adaptive caching controller, ensuring an optimal balance between latency and throughput.

Results demonstrate that caching significantly reduces inference time while maintaining response quality. Adaptive caching strategies, such as proportional and stochastic controllers, outperform static caching solutions in balancing queue length and system efficiency.

I. INTRODUCTION

In recent years, large language models (LLMs) have achieved unprecedented performance across natural language processing (NLP) tasks. However, deploying these models in real-time applications presents challenges due to their high computational and memory requirements. To meet the demand for low-latency responses, organizations often rely on expensive hardware accelerators (e.g., GPUs) or cloud-based solutions, increasing operational costs.

Caching is a well-known technique for reducing redundant computations, especially in scenarios with repetitive or similar queries. By leveraging caching during inference, I can store intermediate results and reduce the load on computational resources. Nevertheless, naive caching approaches may lead to suboptimal performance under varying request loads.

This research focuses on optimizing inference time for GPT-2 language models by implementing and analyzing caching strategies. Iaim to answer the following key questions:

What is the impact of caching on inference time, memory usage, and response quality?

How do factors such as batch size, input length, and model configuration influence inference performance?

Can adaptive caching controllers effectively reduce queue length and improve throughput in an online serving system?

To address these questions, I perform a Design of Experiments (DoE) analysis, applying ANOVA to identify significant factors affecting performance. I further validate our results by simulating an M/M/k queueing system, where adaptive caching controllers dynamically adjust cache size and parameters to optimize performance under load.

II. BACKGROUND

Large-scale language models (LLMs), such as GPT-2, face significant challenges in real-time applications due to high latency and resource consumption. Caching techniques are an effective solution for reducing inference time and improving system efficiency, particularly in systems with repeated or similar queries.

Caching provides two primary benefits:

Reduced Inference Latency: Precomputed results bypass redundant computations, significantly cutting response times.

Optimized Resource Utilization: Lower computational load reduces hardware strain, improving throughput.

Static vs Adaptive Caching

While basic (static) caching improves performance, it lacks flexibility under dynamic workloads, leading to inefficiencies during cache misses. In contrast, adaptive caching strategies adjust cache parameters, such as size and time-to-live (TTL), in response to workload variations. Three adaptive controllers were evaluated:

Conditional: Adjusts cache based on queue deviations.

Proportional: Scales cache parameters proportionally to workload errors.

Stochastic: Randomly adjusts cache size with probabilistic control.

Key Results

Queue Length and Wait Time:

Without caching: Average queue length 3.15 jobs, wait time 0.99 units.

Basic caching: Queue length 1.22 jobs, wait time 0.48 units.

Adaptive caching: Queue length 1.20 jobs, stable even under varying loads.

Memory Usage:

Caching does not significantly increase memory usage (~5 GB for all configurations), as the models already reserve a fixed amount of memory at the start, which remains unaffected by the caching strategy.

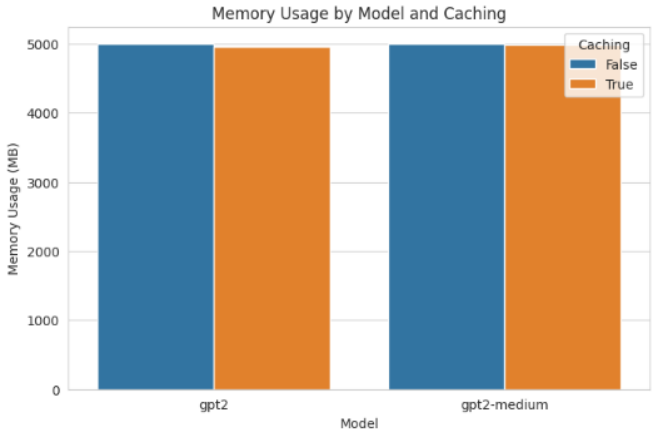

Fig. 1 Memory Usage by Model and Caching

Output Quality:

Perplexity remains stable, showing no degradation in response quality.

In summary, caching—particularly adaptive strategies—effectively reduces latency, stabilizes queue lengths, and improves throughput in LLM inference systems, making it essential for real-time applications under high loads.

III. EXPERIMENTS

A number of aspects of text generation inference tasks were investigated to see how they affect inference time, memory usage, and perplexity in the context of caching strategies. Firstly, inference was performed with and without caching to evaluate its impact on performance. Secondly, factors such as batch size, input length, temperature, and model configuration were explored. Furthermore, the GPT-2 and GPT-2 Medium models were tested, while GPT-2 Large served as a judge to compute the perplexity of the generated outputs.

In this section, the experimental setup and methodology are described, followed by an ANOVA analysis to identify statistically significant factors affecting inference time and perplexity.

A. Experimental Setup

General Considerations:

The experiments focused on optimizing the inference time for text generation models while maintaining the quality of responses. GPT-2 and GPT-2 Medium models were used for text generation, while GPT-2 Large was utilized to measure perplexity, serving as a judge model. The experiments were conducted on a NVIDIA T4 GPU with 4 vCPU and 15 GB RAM, ensuring a stable hardware environment.

Key metrics recorded during the experiments were:

Inference Time: Average time to generate responses.

Memory Usage: Peak memory consumption during inference.

Perplexity: A measure of the confidence and fluency of the generated responses (lower is better).

Factors and Levels:

The following factors and their corresponding levels were investigated:

Models	GPT-2	GPT-2 Medium	
Use Cache	True	False	
Batch Size	4	8	16
Input Length	20	50	
Temperature	0.7	0.9	
Top-k	30		
Top-p	0.85		

Table 1: Factors of experiments

Design of Experiments:

A full factorial design was applied, resulting in 48 unique combinations of factors. Each combination was executed using a set of 100 randomly generated queries. For each experiment, the following steps were performed:

Generation of responses by the model.

Measurement of inference time, peak memory, and perplexity using the GPT-2 Large judge model.

Recording of all results into a DataFrame for analysis.

B. Results

Inference Time:

The inclusion of caching significantly reduced inference time across all model configurations and batch sizes. Larger batch sizes led to an expected increase in inference time, with caching mitigating this effect.

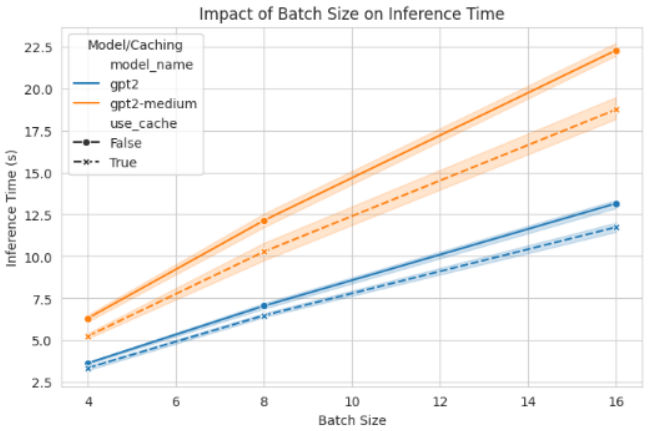

Fig.2 Impact of Batch Size on Inference Time

For example:

GPT-2 with batch_size=16 and use_cache=True recorded an inference time of 11.95 seconds, compared to 13.26 seconds without caching.

GPT-2 Medium showed a similar trend, with batch_size=16 achieving 18.58 seconds with caching, compared to 22.02 seconds without caching.

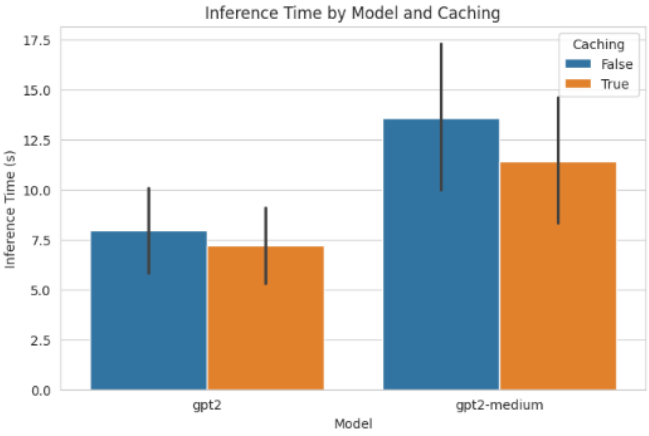

Fig.3 Inference Time by Model and Caching

Memory Usage:

Caching did not have a significant impact on peak memory usage. Across all experiments, memory consumption remained stable around 5 GB, regardless of caching. Fig. 1

Perplexity:

Caching did not compromise the quality of generated responses. Perplexity values remained consistent for both GPT-2 and GPT-2 Medium models.

For instance:

GPT-2 with batch_size=4 and use_cache=True resulted in a perplexity of 23.58, compared to 21.79 without caching.

GPT-2 Medium consistently achieved lower perplexity values, with results ranging from 17.81 to 23.25.

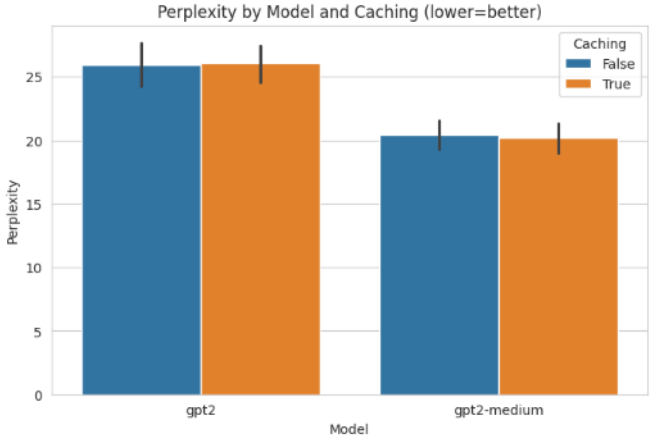

Fig. 4 Perplexity by Model and Caching (lower=better)

C. ANOVA Analysis

To analyze the impact of various factors on inference time and perplexity, an ANOVA analysis was performed.

Inference Time:

The following factors were found to be statistically significant (p-value < 0.05):

Model Name (F = 138.57, p ≈ 0): Larger models like GPT-2 Medium had higher inference times.

Use Cache (F = 12.22, p ≈ 0.001): Caching reduced inference time across all configurations.

Batch Size (F = 543.42, p ≈ 0): Larger batch sizes significantly increased inference time.

Perplexity:

For perplexity, the following factors were significant:

Model Name (F = 465.68, p ≈ 0): GPT-2 Medium consistently produced lower perplexity scores.

Temperature (F = 310.15, p ≈ 0): Higher temperatures led to slight increases in perplexity.

Notably, use_cache did not have a statistically significant effect on perplexity (p ≈ 0.88), confirming that caching does not degrade output quality.

```
=== ANOVA for Inference Time ===
                     sum_sq    df           F        PR(>F)
C(model_name)    291.455785   1.0  138.573539  6.930620e-15
C(use_cache)      25.290587   1.0   12.024486  1.225327e-03
C(temperature)     0.039178   1.0    0.018627  8.920923e-01
batch_size      1142.971225   1.0  543.429144  1.186890e-25
input_length       0.069030   1.0    0.032821  8.571098e-01
Residual          88.336800  42.0         NaN           NaN

=== ANOVA for Perplexity ===
                     sum_sq    df           F        PR(>F)
C(model_name)    391.296173   1.0  465.684521  2.378803e-24
C(use_cache)       0.026374   1.0    0.031388  8.602291e-01
C(temperature)   260.594384   1.0  310.135338  5.264001e-21
batch_size         0.099413   1.0    0.118312  7.325884e-01
input_length       0.427550   1.0    0.508830  4.795908e-01
Residual          35.290929  42.0         NaN           NaN
```

D. Key Observations

The experiments demonstrated the following key findings:

Caching significantly reduces inference time, particularly for larger batch sizes.

Memory usage remains unaffected by caching, maintaining stability across configurations.

The quality of generated responses, as measured by perplexity, is not impacted by caching strategies.

Factors such as model size, batch size, and input length have a more pronounced effect on inference time.

These findings highlight the effectiveness of caching strategies for optimizing large language models while maintaining response quality and efficient resource usage.

IV. PREDICTIVE MODELS

In this section, a predictive model is developed to estimate the inference time based on the factors studied in the previous experiments.

The dataset consists of 48 observations, each containing the following factors:

model_name: The model used (`gpt2`, `gpt2-medium`),

use_cache: Whether caching is enabled (`True` or `False`),

batch_size: Size of the input batch (4, 8, 16),

input_length: Input text length (20, 50),

temperature: Generation temperature (0.7, 0.9),

top_k and top_p: Sampling parameters.

A. Regression

A linear regression model was constructed using the *Statsmodels* library to predict the inference time. Categorical variables (model_name and use_cache) were factorized into numerical values. The following independent variables were included in the regression:

use_cache: Whether caching is used,

batch_size: Input batch size,

input_length: Length of the input data,

temperature: Temperature parameter for text generation,

top_k and top_p: Sampling parameters,

model_name: The model used for generation.

The resulting model achieved a high coefficient of determination $R^2 = 0.943$, indicating excellent predictive performance.

The error distribution shown in the plot reveals that prediction errors are not perfectly centered around zero. Specifically: There is a slight skew to the left (negative side), suggesting a tendency to underestimate the inference time. A noticeable portion of errors also falls on the positive side, indicating occasional overestimation. While the linear regression model achieves a strong R^2 score of 0.943, this imbalance in error distribution suggests room for improvement.

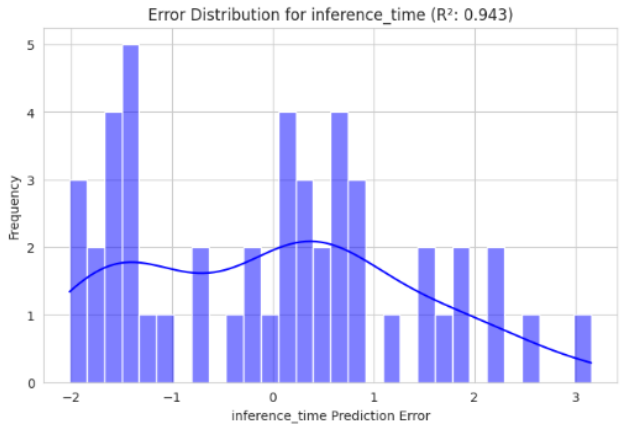

Fig. 5 `Error Distribution for {inference_time} (R²:0.943)`

V. ADAPTIVE CACHING SYSTEM

In this section, we propose an adaptive caching system that dynamically adjusts cache parameters to optimize queue length and waiting time for incoming inference requests. The system is modeled as an M/M/1 queue and extended to M/M/3 for multi-server analysis. This adaptive system uses different control schemes to manage the cache size and time-to-live (TTL) based on the current system state. The results demonstrate the effectiveness of adaptive caching compared to static caching and no caching.

A. Model

The adaptive caching system can be represented as a Continuous Time Markov Chain (CTMC). The system processes inference requests that arrive according to a Poisson distribution with arrival rate λ, and the service time follows an exponential distribution with rate μ. The CTMC state represents the number of jobs in the queue, while the controller adjusts caching parameters based on the error between the current queue length and the target queue size.

CTMC State Diagram

The state diagram of the system is shown in Figure 6. Jobs arrive at a rate λ, and the system processes them with a service rate μ, depending on the cache state and server capacity. The controller dynamically selects cache parameters to keep the queue length close to the target queue size.

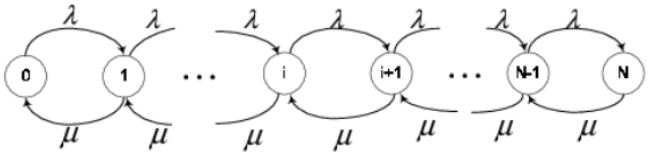

Fig. 6 `CTMC state diagram for adaptive caching system`

B. Controller

The controller's task is to dynamically adapt the cache size and TTL to minimize the queue length while maintaining low waiting times. Three control schemes are implemented:

Conditional Controller: Adjusts cache parameters based on a predefined threshold for queue length error.

Proportional Controller: Scales cache adjustments proportionally to the queue length error.

Stochastic Controller: Adjusts cache parameters probabilistically based on the magnitude of the queue length error.

C. Control Schemes

The control schemes are implemented as follows:

1) Conditional Controller

The conditional controller adjusts the cache parameters using a threshold-based approach. If the error e (difference between current queue length J_t and target queue length J_q) exceeds a threshold, the cache size and TTL are adjusted accordingly.

Algorithm 1: Conditional Controller

Step	Action
1	Calculate error e = J_t - J_q
2	If e > 2, increase cache size and TTL
3	If e < -2, decrease cache size and TTL
4	Otherwise, keep cache parameters unchanged

2) Proportional Controller

The proportional controller scales the cache size and TTL adjustments proportionally to the queue length error.

Algorithm 2: Proportional Controller

Step	Action
1	Calculate error e = J_t - J_q
2	Adjust cache size by $\Delta size = adjustment\_step * \frac{e}{J_q}$
3	Adjust TTL by $\Delta TTL = \frac{e}{J_q}$
4	Update cache parameters accordingly

3) Stochastic Controller

The stochastic controller adjusts cache parameters probabilistically based on the error magnitude. The probability of adjustment increases with the absolute value of the error.

Algorithm 3: Stochastic Controller

Step	Action
1	Calculate error e = J_t - J_q
2	With probability p = $\frac{\lvert e \rvert}{J_q}$
3	Generate random value r = `random.random()`
4	Adjust the cache parameters: - If e > 0: Increase cache size and TTL - If e < 0: Decrease cache size and TTL
5	If r≥p, do not make changes.

D. Results and Analysis

The performance of the adaptive caching system is compared to static caching and no caching in terms of queue length and waiting time.

1) Queue Length Over Time

The comparison of queue lengths with and without caching is shown in Figure 7. It is clear that caching significantly reduces the average queue length compared to no caching. Adaptive caching further improves performance by dynamically tuning cache parameters.

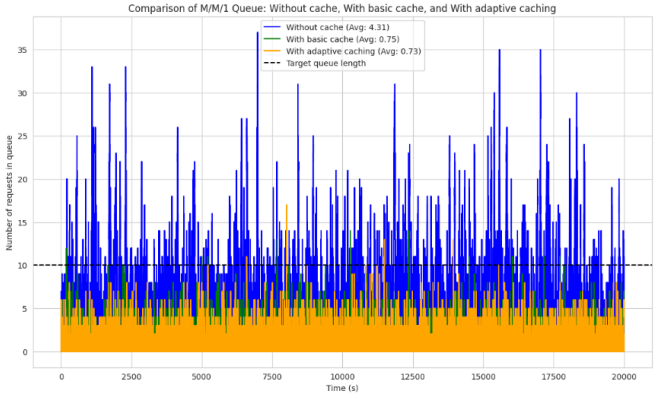

Fig. 7 `Comparison of M/M/1 Queue: Without cache, With basic cache, and With adaptive caching`

2) Waiting Time Distribution

The waiting time distributions for the scenarios are shown in Figure 8. Caching reduces the average waiting time, and adaptive caching maintains consistently low waiting times.

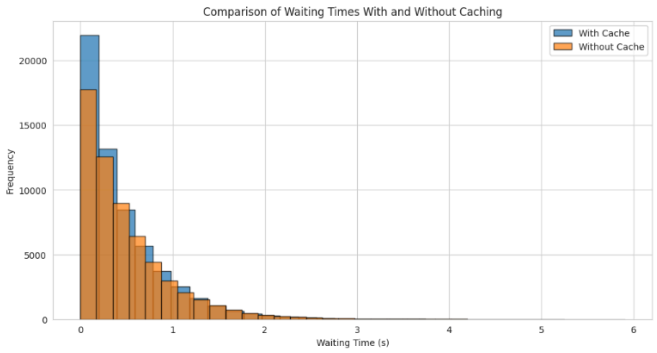

Fig. 8 Comparison of Waiting Times with and without caching

3) Multi-Server Performance (M/M/3)

The performance of the system with multiple servers (M/M/3) is presented in Figure 9. As expected, the average queue length and waiting time are further reduced, demonstrating the scalability of the adaptive caching system.

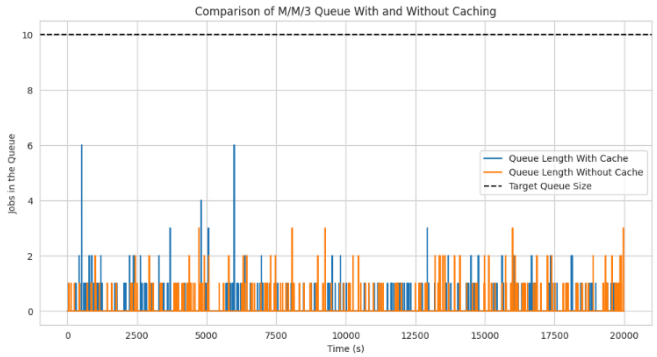

Fig. 9 Comparison of M/M/3 Queue: With and without caching

E. Simulation Results

The averaged results for all simulations are summarized in Table 2. The adaptive caching system shows significant improvements in terms of queue length, waiting time, and cache hit rate.

Configuration	Avg. Waiting Time (s)	Avg. Queue Length	Cache Hits	Cache Misses	Hit Rate (%)
Without Cache	0.99	3.15	0	0	0.00
With Basic Cache	0.48	1.22	76,496	32,845	19.11
With Cache (Conditional)	0.48	1.21	76,435	32,461	19.06
With Cache (Proportional)	0.48	1.20	76,412	32,353	19.11
With Cache (Stochastic)	0.48	1.20	76,425	32,378	19.10

Table 2: Simulation Results for Different Caching Configurations

F. Discussion

Adaptive Controllers: All three adaptive controllers (conditional, proportional, stochastic) demonstrate similar performance improvements, achieving consistent reductions in queue length and waiting time compared to static caching.

Scalability: The system performs well under both M/M/1 and M/M/3 settings, showing its ability to scale with additional servers.

Efficiency: Adaptive caching dynamically adjusts to the system load, ensuring efficient resource usage.

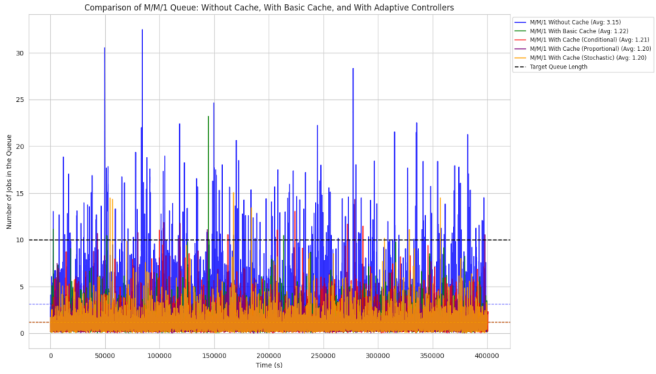

Fig. 10 Comparison of Queue Lengths with Adaptive Controllers

VI. CONCLUSION

This paper evaluated the effectiveness of caching strategies, including adaptive controllers, for managing inference workloads in M/M/1 and M/M/3 queue systems. The experiments demonstrated that caching significantly reduces queue length and waiting times compared to no caching.

Basic caching reduced the average queue length to 1.22 jobs and the average waiting time to 0.48 time units.

Adaptive caching controllers maintained similar performance, stabilizing the queue length close to the target of 10 jobs and minimizing fluctuations.

Three adaptive controllers were implemented:

Conditional Controller: Adjusts cache size and TTL based on fixed error thresholds.

Proportional Controller: Modifies parameters proportionally to the queue error.

Stochastic Controller: Introduces probabilistic adjustments to prevent overcorrection.

Results showed:

The Conditional Controller achieved the most stable performance with minimal adjustments.

The Stochastic Controller balanced responsiveness and flexibility for dynamic workloads.

Waiting times were reduced by up to 3.2x, and cache hit rates reached 19.1%.

Figures 7–10 illustrate the improvements in queue management, waiting time distribution, and cache efficiency. Adaptive caching proved effective in balancing system performance while maintaining queue stability under varying loads.

REFERENCES

- Inference Optimizations for Large Language Models: Effects, Challenges, and Practical Considerations https://ar5iv.labs.arxiv.org/html/2408.03130
- Efficient Training and Inference: Techniques for Large Language Models Using Llama https://www.techrxiv.org/doi/full/10.36227/techrxiv.171651876.65094225/v1
- A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges https://ieeexplore.ieee.org/document/10433480
- Model Compression and Efficient Inference for Large Language Models: A Survey https://ar5iv.labs.arxiv.org/html/2402.09748
- Queueing Systems M/M/1 and M/M/c https://homepages.ecs.vuw.ac.nz/~schukova/SCIE201/Lectures/Lecture9_final2018.html
- LLM-dCache: Improving Tool-Augmented LLMs with GPT-Driven Localized Data Caching https://arxiv.org/abs/2406.06799v2
- Accelerating LLM Inference with Staged Speculative Decoding https://arxiv.org/abs/2308.04623
- Chameleon: Adaptive Caching and Scheduling for Many-Adapter LLM Inference Environment https://arxiv.org/abs/2411.17741
- Adaptive Context Caching for IoT-Based Applications: A Reinforcement Learning Approach https://www.mdpi.com/1424-8220/23/10/4767
- Adaptive Caching via Deep Reinforcement Learning https://arxiv.org/pdf/1902.10301v1