International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

OPTIMIZING LLM PERFORMANCE THROUGHCI /CD PIPELINES IN CLOUD-BASED ENVIRONMENTS

Reena Chandra[1], Kishore Ranjan[2], Karan Lulla[3]

[1]Independent Researcher, San Francisco, CA, USA.

Email: reenachandra11@gmail.com - ORCID: 0009-0001-8061-1084

[2]Independent Researcher, Trumbull, CT, USA.

Email: kishoreranjan@gmail.com- ORCID: 0009-0001-5372-9490

[3]Independent Researcher, San Francisco, CA, USA.

Email: kvlulla16@gmail.com - ORCID: 0009-0007-7491-4138

Abstract

The deployment of large language models (LLMs) in cloud environments presents significant challenges, particularly due to their high computational demands, latency, memory consumption, and the lack of automated and reproducible workflows. The need for efficient, low-cost, reproducible deployment strategies has become critical as LLMs continue to scale and become integral to enterprise and research systems. Traditional manual deployment methods often result in performance instability and hinder operational scalability. To address these issues, this study explores the integration of CI/CD (Continuous Integration/Continuous Deployment) pipelines within Python-based cloud environments as a lightweight alternative for automating model benchmarking and inference tracking. Using the Open LLM Performance Benchmark dataset, which includes metrics such as model size, benchmark scores (e.g., ARC, MMLU, HellaSwag, TruthfulQA), latency, and memory usage, we evaluate a diverse set of public models, including DistilGPT -2, TinyLlama, GPT-Neo-125M, Falcon-rw-1b, and others. All experiments are conducted within Google Colab to simulate low-infrastructure environments. The proposed CI/CD workflow incorporates automated prompt generation, inference execution, latency and memory profiling, and structured logging. Additionally, version control is simulated using DVC-style file hashes and experiment tracking through MLflow. Key findings highlight a clear tradeoff between model size, performance, and cost. Smaller models, such as Tiny-GPT2, demonstrate superior latency but reduced benchmark scores, whereas larger models, like Falcon-rw-1b, yield higher accuracy at the expense of increased memory and inference time. The CI/CD pipeline improved reproducibility, execution traceability, and scalability. These results underscore the potential of lightweight CI/CD frameworks to streamline LLM deployment for teams operating under resource constraints.

Keywords: LLM deployment, CI/CD pipelines, Google Colab, inference benchmarking, MLOps, model reproducibility, cloud-based NLP

Received: July 08, 2025 ; Published: September 19, 2025

1 Introduction

By implementing large language models (LLMs), various sectors, academics, and open-source projects have significantly influenced changes in natural language processing (NLP) [1]. Conversational agents, tools that summarise information, and intelligent code assistants utilise LLMs today. As organisations grow and take on more challenges, these models have become necessary for significant projects. At the same time, this growth gives rise to various difficulties during deployment [2]. The most critical factors are Inference latency, the hardware level used, and the lack of deployable workflows that can be reused. As the cloud's complexity increases, ensuring the right resource balance, high performance, and low costs becomes essential [3].

Using manual fail tactics to install LLMs makes it challenging to monitor the LLM's performance, manage its various versions, or perform rollbacks [4]. As a result, research and production environments miss out on replicability and the ability to trace the steps taken in experiments. The suggested approach in this paper is to utilise MLOps, with a focus on incorporating Continuous Integration and Continuous Deployment (CI/CD) to address the problems above. CI/CD pipelines streamline testing, error reporting, version control, and adjustments guided by feedback, enabling them to be performed quickly and automatically.

The study chose Google Colab because it is easy to use, supports free-tier GPUs, and integrates seamlessly with Python. Colab is not meant for actual productions, but its low cost proves useful for testing lightweight CI/CD designs. We concentrate on setting up and testing a CI/CD pipeline using available models from Hugging Face Hub and results from the Open LLM benchmark. The framework uses automated prompt inference, measures speed and memory, keeps a log, and tracks different versions using psutil, tracemalloc, hashing, and MLflow. The objectives shaping the research are mentioned below:

1. Evaluate LLMs using public performance benchmarks.
2. Build and simulate a CI/CD pipeline on Google Colab.
3. Analyse reproducibility, latency, memory, and compute behaviour across models.
4. Report cost-benefit tradeoffs for resource-constrained deployment.

With this, the paper examine whether AI teams with limited resources can effectively utilise CI/CD. In addition to evaluating the reproducibility and performance of various open-source LLMs, we also investigate the relationships between accuracy, resource utilisation, and task completion speed. Next, the cost of using each runtime environment is estimated to determine which one is more affordable: Colab CPU, Colab GPU, or Google Cloud A100. The rest of the document is arranged as follows: a literature review explores existing techniques for using LLMs and MLOps; the methodology section outlines the experimental design and how it was carried out; the results section studies the experimental findings; and lastly, the paper covers what was learned and what can be studied in the future.

Received: July 08, 2025 ; Published: September 19, 2025

2. Literature Review

2.1 LLM Deployment in Production

Integrating large language models into software and machine learning creates both benefits and challenges for these fields [5]. Recently, variants of Falcon and LLaMA, alongside GPT-2, BERT, and GPT-Neo, have demonstrated exceptional capabilities in natural language processing and generation tasks [6]. However, putting these models into practical use in production is not easy. Training on large models with hundreds of millions to billions of parameters affects the time it takes for the data to load, run, and consume memory space [7]. Reducing cloud latency becomes even more challenging when starting up large models, which is a time-consuming process [8]. This makes it difficult for the service to consistently deliver high-quality results, as the outcome depends on many unpredictable variables. For example, with GPT, OpenAI has enhanced its infrastructure by utilising custom runtime kernels, model quantisation, and distributing model components across multiple regions [9]. Google Cloud's BERT API also includes orchestration software, making routing requests more efficient and streamlined. Although Falcon performs very well, deploying it outside dedicated GPUs is challenging due to its heavy resource requirements [10, 11]. They demonstrate that model quality often conflicts with the state of the infrastructure when deploying large language models (LLMs).

2.2 CI/CD and MLOps in ML Lifecycle

Meanwhile, as these deployment issues arise, the MLOps approach is becoming popular to organise the machine learning process [12]. Automated testing, integration, and delivery are the primary reasons MLOps utilises CI/CD frameworks, and these tools have also been widely adopted in traditional software engineering [13, 14]. In machine learning (ML), CI/CD pipelines enable the conduct of multiple experiments, automate their validation, and facilitate the safe application of new models [15]. GitHub Actions allows for the automation of tasks when code or data is updated, and MLflow helps with tracking, numbering models and making code repeatable [16]. Jenkins handles the task of running multi-stage pipelines in many enterprises. However, DVC (Data Version Control) is preferred when developers want to manage data and model artefacts in a Git-like system [17]. As a result, ML specialists can use the code → model → test → deploy → monitor → rollback cycle, which is necessary for handling the uncertain and diversified parts of ML jobs [18]. It is standard practice to utilise CI/CD for A/B tests and canary networks, as well as to detect model changes over time in production-ready AI systems [19]. This is because most of them are set up for organisations with large-scale DevOps resources. Hence, they are incompatible with Google Colab and Jupyter-based notebooks, which are lightweight environments.

2.3 Open LLM Benchmarks

Since generalisation and reasoning ability on various tasks are essential for LLMs, several benchmark datasets and leaderboards have been created to assess their performance [20]. It is also worth noting that the Open LLM Performance Benchmark collates results from

Received: July 08, 2025 ; Published: September 19, 2025

four key benchmarks: ARC, HellaSwag, MMLU, and TruthfulQA. These measures are now the standard way to assess the quality of an LLM. Since GPT-Neo, Falcon and TinyLlama are evaluated using these tests, their scores are often used to validate them before deployment [21, 22]. Nevertheless, these benchmarks do not accurately represent practical situations outside of experiments. The problems tested in the papers are usually measured without regard for actual latency, memory required, dependability of results, or the cost of operation, all of which are crucial when the application is used. It requires considerable effort to reach the point where a product can be produced and sold. Typically, benchmarks overlook whether models function properly in resource-constrained environments, how frequently they deliver consistent performance, and whether they are suitable for integration into continuous integration/continuous deployment (CI/CD) processes [23]. For this reason, remote developers often lack sufficient deployment tips; therefore, infrastructure-aware comparison must be a significant yet rarely utilised check in LLM research.

2.4 Model Comparison Studies

Researchers have attempted to address this issue through model comparison studies, focusing on accuracy, model storage requirements, and prediction speed. Since GPT-2 is straightforward and freely available, it has been extensively studied by numerous researchers who have written about its capabilities in various queries and tasks [24]. DistilGPT2 is often appreciated for requiring 60% less space than the original GPT2, yet achieving only a 5% drop in performance. Since TinyLlama and GPT-Neo are compact and can run them through Hugging Face Transformers, they are widely used [25, 26]. However, only a few studies have investigated real-time inference skills, the average time it takes to run, or whether ML models can be integrated with CI/CD tools. While Hugging Face and EleutherAI have begun to list latency and throughput metrics in their model cards, no easy-to-use and standardised frameworks for evaluating models and simulating the existence of Colab or Vertex AI have yet been established. Moreover, most comparison studies overlook reproducibility, although it plays a significant role in managing different versions and ensuring the auditing of AI-based systems.

Benchmarking and comparing large language models (LLMs) has received significant attention, but little research has examined this topic using modern software development practices, such as MLOps or CI/CD. The literature has an essential gap because current approaches do not assess reproducibility, latency, or cost modelling. To address this issue, this study presents a straightforward approach to benchmarking CI/CD using public metrics, running metrics, and tools that foster reproducibility in the cloud. Since this project utilises open infrastructure, such as Google Colab, for testing LLMs, it introduces a novel technique for evaluating models under limited and straightforward conditions.

3. Methodology

This section explains how the framework was built to test and simulate a lightweight CI/CD pipeline for LLM inference. It includes open-access solutions, databases used by many

Received: July 08, 2025 ; Published: September 19, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

and tools for reproducing results, all working together as a cohesive process run on Google Colab. Instead of using the methodology on actual production, it is designed to simulate events and focus on making everything reproducible, fast and easy to analyse in a limited environment. While collecting essential metrics such as latency and CPU workload, the pipeline features experimental versions and uses simulated DVC to verify each command.

3.1 Research Design

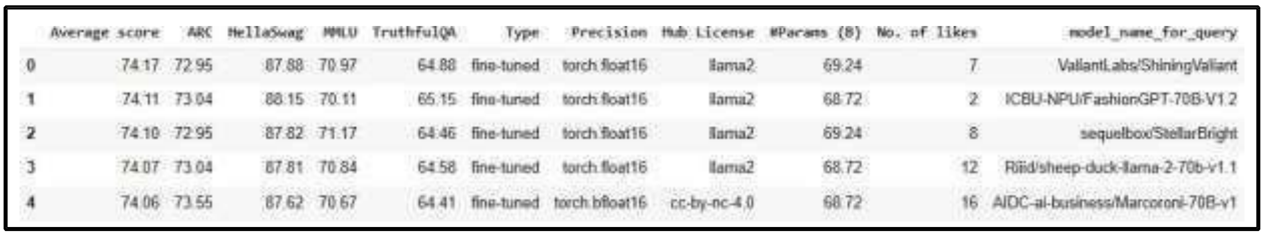
Most of the study employs an experimental simulation technique to evaluate the capabilities of LLMS systematically. All the simulation work was conducted on Google Colab, a cloud-based Python environment that offers GPU and CPU resources for lightweight machine learning tasks. If work relies on continuous integration and deployment, Colab gives you enough control over your experiments to effectively test models and train them.

They decided to set up a Python-built CI/CD framework that performs inference tests using lightweight large language models (LLMs). To avoid exhausting Colab's memory and processing power, the study aimed to analyse the inference performance of various models and collect related metrics without training them. The purpose of the pipeline architecture was to replicate how a model would be tested against multiple inputs in a real CI/CD setup before being made ready for deployment.

Using Python, custom code was written to monitor the script's speed over time.perf_counter, how much memory it used with memory_profiler, the CPU usage with psutil, and to give it a reproducible ID via hashlib. We handled all output in a DataFrame, saved it to CSV and JSON, and ensured versioning using simulated DVC-style hashes. Tracking experiments became possible through MLflow, which saved latency metadata and performance information, as well as utilised memory and model versions.

3.2 Dataset Description

The performance was evaluated using the Open LLM Performance Benchmark, a dataset on Kaggle that contains results from over 1,400 language models on various standard benchmarks [27]. The data includes the names of the models, the number of parameters they contain and scores from ARC, HellaSwag, MMLU and TruthfulQA. The benchmarks assess the model's performance in various areas, including commonsense reasoning, factual knowledge, multitasking, and handling misinformation.



	Average score	ARC	HellaSwag	MMLU	TruthfulQA	Type	Precision	Hub License	#Params (B)	No. of likes	model_name_for_query
0	74.17	72.95	87.88	70.97	64.88	fine-tuned	torch.float16	llama2	69.24	7	ValiantLabs/ShiningValiant
1	74.11	73.04	88.15	70.11	65.15	fine-tuned	torch.float16	llama2	68.72	2	ICBU-NPU/FashionGPT-70B-V1.2
2	74.10	72.95	87.82	71.17	64.46	fine-tuned	torch.float16	llama2	69.24	8	sequelbox/StellarBright
3	74.07	73.04	87.81	70.84	64.58	fine-tuned	torch.float16	llama2	68.72	12	Riiid/sheep-duck-llama-2-70b-v1.1
4	74.06	73.55	87.62	70.67	64.41	fine-tuned	torch.bfloat16	cc-by-nc-4.0	68.72	16	AIDC-ai-business/Marcoroni-70B-v1

Figure 1: Dataset Description

Received: July 08, 2025 ; Published: September 19, 2025

International Journal of Applied Mathematics
Volume 38 No. 2s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Apart from scores on benchmarks, the data also reports average latency, memory requirements and the time taken to load the training model. Using the results from these metrics, we found models that would suit the project and work well with Google Colab. Not all open-source models are labelled the same, so researchers assigned scores to each model using their best knowledge. The primary purpose of preparing the dataset was to use it as a reference. Inference tests were performed separately in Colab, and their performance was observed to compare against the scores reported by the Open LLM Benchmarkers.

3.3 Model Selection Rationale

Three parameters were used to select the five models: they had to be under 2 billion, available to everyone on Hugging Face, and suitable for the memory limits of Colab. The models mentioned were:

- distilgpt2: A distilled version of GPT-2 optimised for speed and lightweight deployment.
- sshleifer/tiny-gpt2: A minimal GPT-2 variant designed for low-memory testing.
- EleutherAI/gpt-neo-125M: A community-developed autoregressive model with open weights.
- tiiuae/falcon-rw-1b: A 1.3B parameter open-weight model known for strong benchmark scores.
- PY007/TinyLlama-1.1 B-Chat-v0.1: A LLaMA-based compact model designed for chat applications.

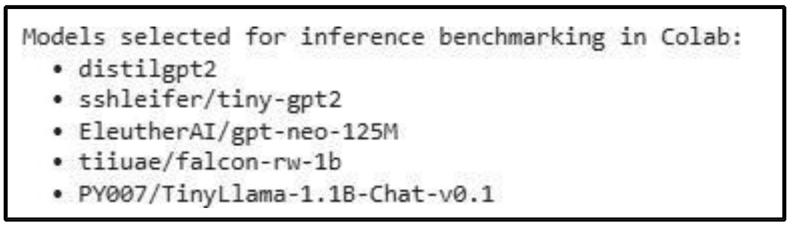

```
Models selected for inference benchmarking in Colab:
  • distilgpt2
  • sshleifer/tiny-gpt2
  • EleutherAI/gpt-neo-125M
  • tiiuae/falcon-rw-1b
  • PY007/TinyLlama-1.1B-Chat-v0.1
```

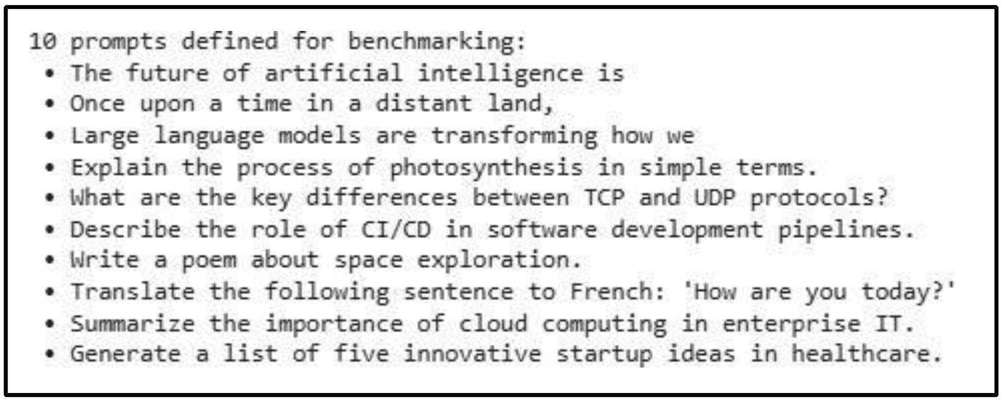
Figure 2: Model Selected

They differ in their performance, the amount of memory needed, and the time each model takes to process. The Hugging Face Transformers library enabled the efficient loading and use of all models. The models we included allow for comparing results in a Colab notebook without straining the available computing resources.

3.4 Benchmark Prompt Design

Ten various prompts were created to test the speed and efficiency of LLMs in real-world applications. The questions were designed to match several areas of LLM work, including telling stories, explaining, translating, summarising, factual question and answer, and creative writing. Examples include:

- “The future of artificial intelligence is...”
- “Explain the process of photosynthesis in simple terms.”
- “Translate the following sentence to French: ‘How are you today?’”
- “What are the key differences between TCP and UDP protocols?”
- “Write a poem about space exploration.”



```
10 prompts defined for benchmarking:
 • The future of artificial intelligence is
 • Once upon a time in a distant land,
 • Large language models are transforming how we
 • Explain the process of photosynthesis in simple terms.
 • What are the key differences between TCP and UDP protocols?
 • Describe the role of CI/CD in software development pipelines.
 • Write a poem about space exploration.
 • Translate the following sentence to French: 'How are you today?'
 • Summarize the importance of cloud computing in enterprise IT.
 • Generate a list of five innovative startup ideas in healthcare.
```

Figure 3: Benchmark Prompt

The various prompts allowed each model to be evaluated with different complex inputs and tokens. The CI/CD pipeline could also assess whether the latency remained consistent for other prompts. The prompts were tested individually on the five models, and the results, along with the corresponding metrics, were saved.

3.5 CI/CD Pipeline Design

The purpose of the CI/CD simulation was to replicate the usual steps involved in an ML pipeline. When using a traditional software setup, a CI/CD system monitors code commits, executes tests, verifies the output, and deploys the software to production or rolls back the changes if issues arise. The design was implemented in the Colab system using Python programming.

The primary step emulated a signal that can activate GitHub Actions. Accurate log files were generated for each step in the script, from raw data to prediction, by processing the data, applying a model, validating the results, and saving the necessary numbers for metric comparison. All outputs from the inference results were checked for accurate types and to ensure they were not null. Whenever an inference was flawed, it was identified as a CI/CD test that had failed. Hashlib.md5 was used to generate hashes for DVC-like files, which were formed by combining the model's name, the prompt, and the output. Whenever a file was tracked, its hash was added to the log for easy tracking. Although unable to support DVC in the specified environment, this code demonstrated the snapshot versioning concept. Additionally, MLflow was utilised to record experiment data, resulting in the creation of a new experiment named "LLM_CI_CD_Benchmark." Latency, memory usage, CPU load, and benchmark score

Received: July 08, 2025 ; Published: September 19, 2025

were designated as MLflow metrics for each model, and the inference logs for each model were stored as artefacts. This made it possible to check the results after finishing an experiment and to reload an experiment run for review.

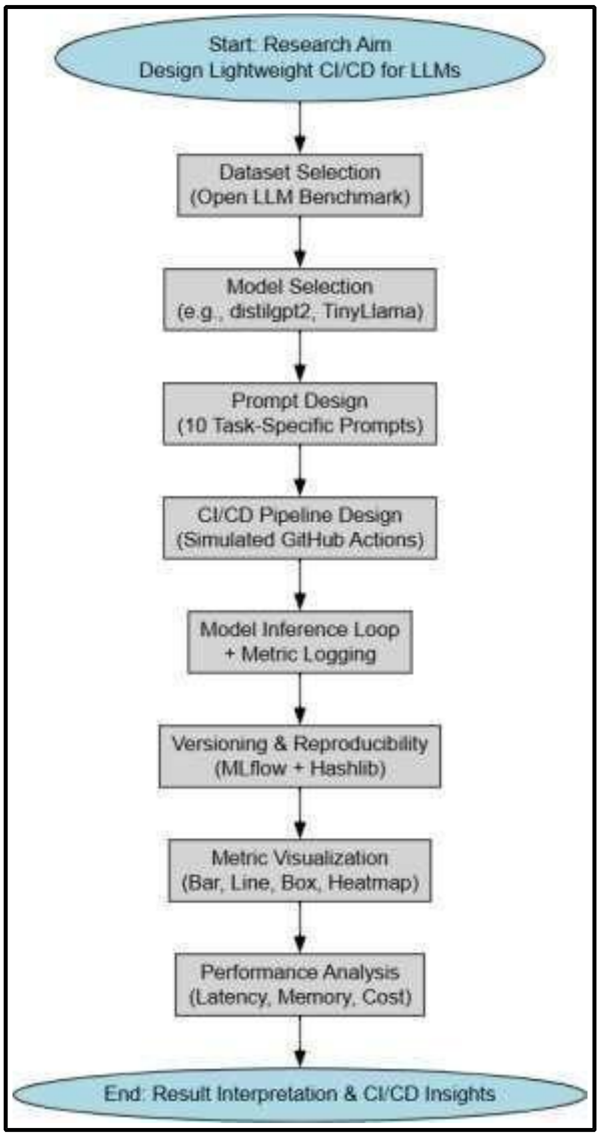


Figure 4: Methodology Flow Diagram

It outlines how continuous integration and delivery evaluate model file deployment in cloud settings with limited resources. It offers automated, trackable features and usable and verifiable guidelines for developing AI-based systems.

4. Implementation

All benchmarking functions were implemented using Google Colab because it offers a convenient and flexible cloud-native CI/CD computing environment. The area was prepared at the start, and the necessary tools were integrated and downloaded, including transformers, torch, psutil, memory_profiler, matplotlib, and seaborn, to support the analysis using Python

Received: July 08, 2025 ; Published: September 19, 2025

libraries. They were picked because they are small and easy to set up in any company's notebook environment. To ensure every model-prompt combination is evaluated similarly, we created specific utility functions for handling prompts and logging the tests.

The primary feature of the pipeline was an inference engine that processed each selected model with ten predefined sets of prompts. In each iteration, the pipeline used Python's time.perf_counter to measure inference latency, memory_usage to check memory usage, and psutil.cpu_percent to determine the percentage of the CPU in use. Apart from evaluating numeric values, the output of every inference was examined to ensure it was formatted correctly and appeared as expected, mimicking a central test assertion to verify that the model responds appropriately. All successful inferences, along with their relevant performance and information, were recorded in a results table. By generating a hash using hashlib.md5 from Python, all the information about the model, input prompt, and output can be stored in a way that resembles DVC.

The study relied on visualising the data to make performance logs understandable. The visualisation engine produced six varieties of plots. It was easy to spot the difference in inference speed by looking at the average latency for each model on the bar chart. It demonstrated the average memory required by each model to determine which models fit best on the given hardware. The boxplot was used to determine if each model could perform consistently when the amount of data was increased or decreased. The scatterplot enables us to visualise the relationship between the model's performance and its speed. A correlation heatmap was developed to check the relationships between latency, memory, CPU usage and benchmark points. Finally, the data was presented in a bar chart to outline cost estimations for Colab CPU, Colab GPU, and GCP A100 GPU, using figures from the required runtime and the public price list.

Two main features were implemented to allow the results to be repeated. First, each inference was assigned a unique hash as a fingerprint for tracking the DVC approach. With these hashes, one can identify specific model-prompt interactions and easily examine or revisit them. It was also added to MLflow, which is widely adopted for experiment tracking. MLflow launched a brand-new experiment called "LLM_CI_CD_Benchmark" and recorded everything needed for the results, both the models and the CSV logs. Colab could model a lightweight CI/CD pipeline with all these details, enabling automated testing, logging, and other similar benefits. This would appeal to models that are understood in real-world application scenarios with limited resources.

Received: July 08, 2025 ; Published: September 19, 2025

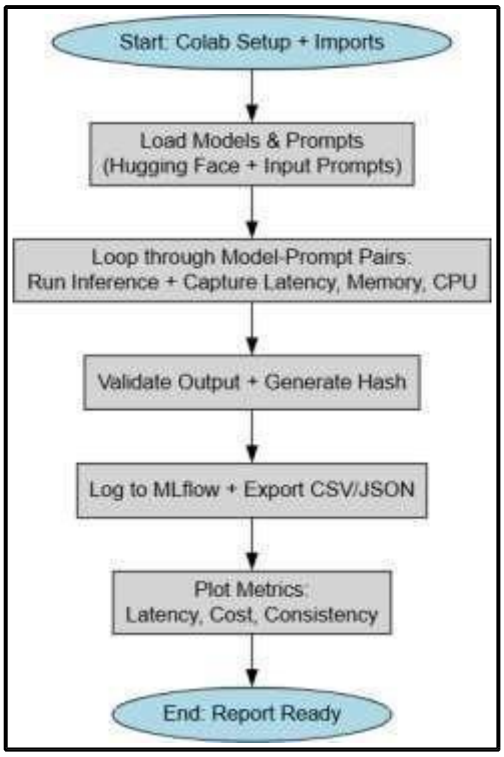

Start: Colab Setup + Imports
Load Models & Prompts (Hugging Face + Input Prompts)
Loop through Model-Prompt Pairs: Run Inference + Capture Latency, Memory, CPU
Validate Output + Generate Hash
Log to MLflow + Export CSV/JSON
Plot Metrics: Latency, Cost, Consistency
End: Report Ready

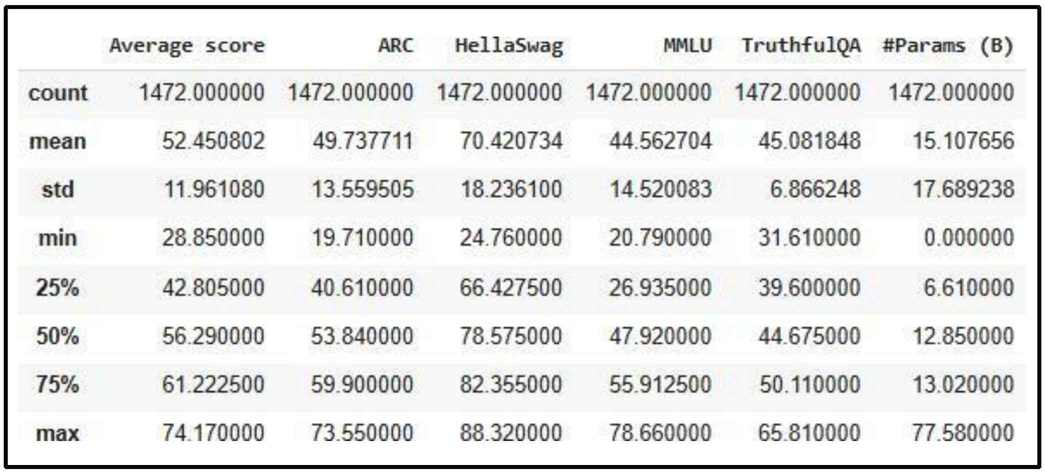
Figure 5: Implementation Design

5. Results and Analysis

5.1 Exploratory Data Analysis (EDA)

An extensive analysis of the dataset was conducted using exploratory data analysis (EDA) before setting up the CI/CD pipeline. Within this dataset are 1,472 large language models, each rated on evaluations for ARC, HellaSwag, MMLU, and TruthfulQA, along with data on their size, precision type, and framework. The average score across all benchmarks provides an overall performance indicator. The first statistics showed that results varied greatly, as average scores in this field went from 28.85 to 74.17, with an approximate average of 52.45. It is worth noting that HellaSwag performed better, with a mean score of 70.42, whereas TruthfulQA presented the widest variations in model difficulty, with a mean of 45.08.

Received: July 08, 2025 ; Published: September 19, 2025



	Average score	ARC	HellaSwag	MMLU	TruthfulQA	#Params (B)
count	1472.000000	1472.000000	1472.000000	1472.000000	1472.000000	1472.000000
mean	52.450802	49.737711	70.420734	44.562704	45.081848	15.107656
std	11.961080	13.559505	18.236100	14.520083	6.866248	17.689238
min	28.850000	19.710000	24.760000	20.790000	31.610000	0.000000
25%	42.805000	40.610000	66.427500	26.935000	39.600000	6.610000
50%	56.290000	53.840000	78.575000	47.920000	44.675000	12.850000
75%	61.222500	59.900000	82.355000	55.912500	50.110000	13.020000
max	74.170000	73.550000	88.320000	78.660000	65.810000	77.580000

Figure 6: Descriptive Analysis

There were more models with fewer than 13 billion parameters, with two secondary groups identified for models with 30 billion and 70 billion parameters, following the pattern observed in open-source large language models (LLMs). On a scatterplot of these measures, Pearson's correlation revealed a positive relationship (~0.57), but also highlighted instances where fewer parameters led to better scores. As seen on the boxplots, MMLU and ARC had larger interquartile ranges, likely caused by their extensive question areas.

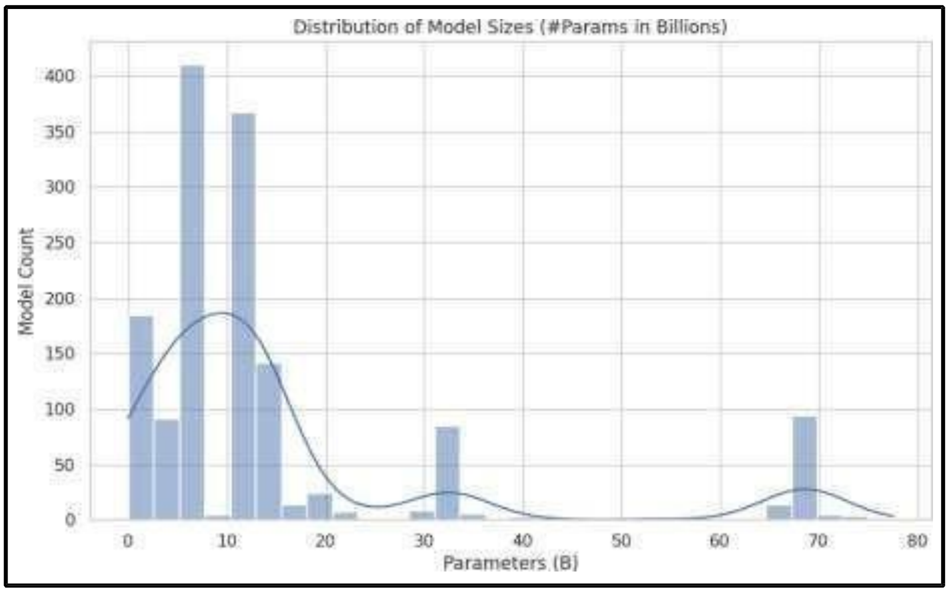


Figure 7: Distribution of Model Sizes

Received: July 08, 2025 ; Published: September 19, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

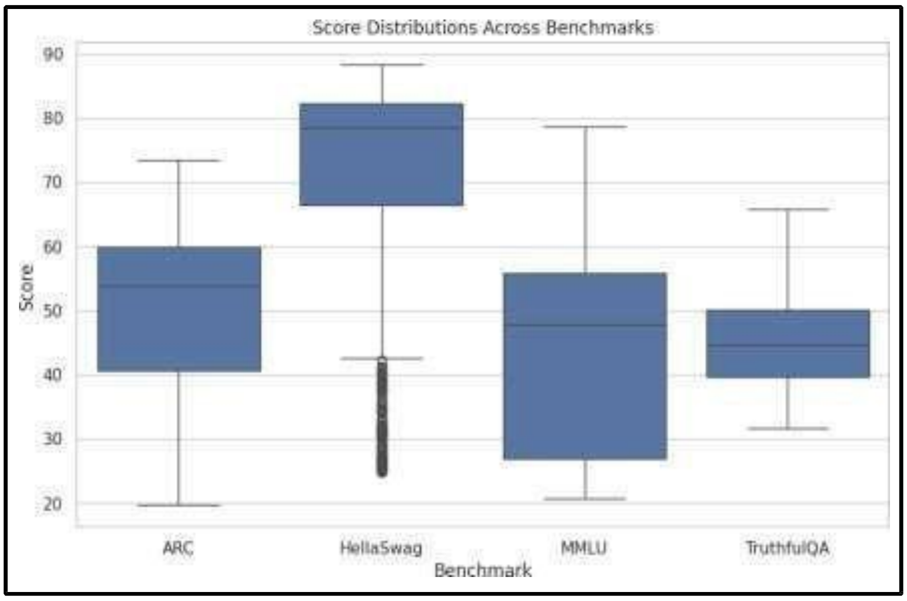


Figure 8: Box Plot of Score Distribution

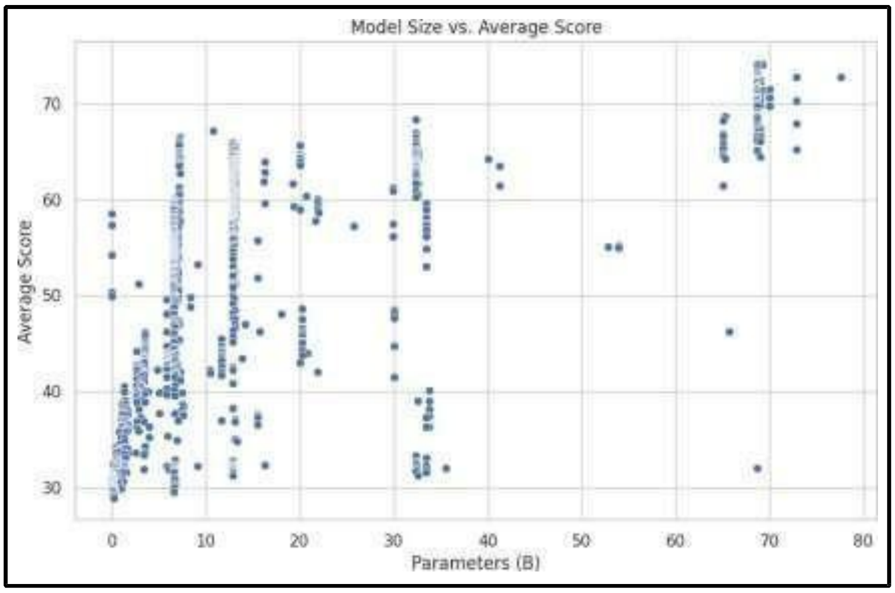


Figure 9: Scatter Plot of Model Size vs Average Score

Moreover, the correlation heatmap indicates that different benchmark scores are closely related (for example, 0.92 for ARC and MMLU), demonstrating the reliability of the benchmark. At the end of the EDA, five different models were selected for testing in CI/CD, all of which had fewer than 2 billion parameters, enabling them to run on Colab. They are distilgpt2, sshleifer/tiny-gpt2, gpt-neo-125M, tiiuae/falcon-rw-1b and TinyLlama-1.1B. They are chosen to ensure a good tradeoff between quick performance testing and leading results, providing a good choice for pipeline modelling and analysis.

Received: July 08, 2025 ; Published: September 19, 2025

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

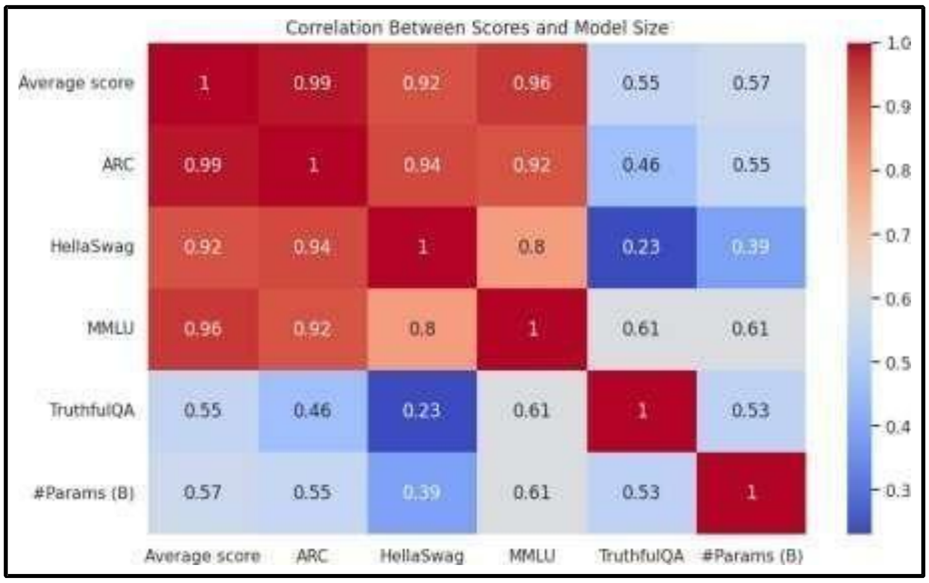
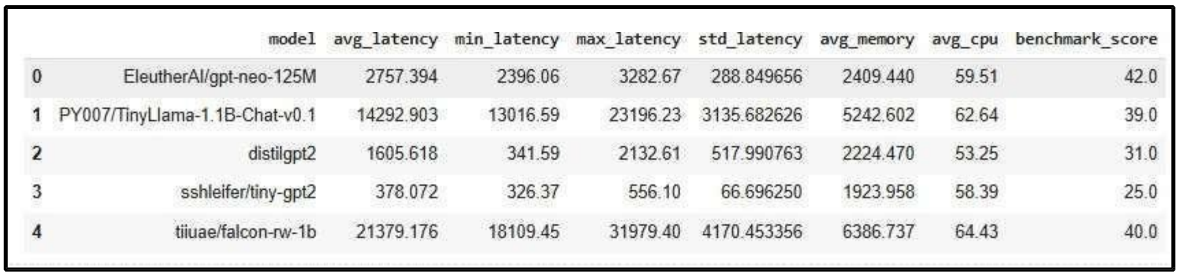


Figure 10: Correlation Heatmap

5.2 Execution Time Summary

Results from latency measurements demonstrated that different models have different inference efficiencies. EleutherAI/GPT-Neo-125 M had an average response time of approximately 2.75 seconds. A similar increase in process time was observed with TinyLlama-1.1 B-Chat-v0.1, which took over 14 seconds to respond on average due to the larger requirements of newer model versions. Its latency of approximately 1606 ms confirms that DistilGPT2 is ideal for lightweight applications that require low latency. Among all the models, the sshleifer/tiny-gpt2 system ran the fastest and had the shortest average latency of only 378 ms. The Falcon-RW-1B may be well-known for its performance, yet its average latency of over 21 seconds suggests that it may not be suitable for real-time inference in the cloud. The evidence also comes from the stable relationship between the minimum and maximum latency, as bigger models tend to be less stable and have a wider variation margin.



	model	avg_latency	min_latency	max_latency	std_latency	avg_memory	avg_cpu	benchmark_score
0	EleutherAI/gpt-neo-125M	2757.394	2396.06	3282.67	288.849656	2409.440	59.51	42.0
1	PY007/TinyLlama-1.1B-Chat-v0.1	14292.903	13016.59	23196.23	3135.682626	5242.602	62.64	39.0
2	distilgpt2	1605.618	341.59	2132.61	517.990763	2224.470	53.25	31.0
3	sshleifer/tiny-gpt2	378.072	326.37	556.10	66.696250	1923.958	58.39	25.0
4	tiiuae/falcon-rw-1b	21379.176	18109.45	31979.40	4170.453356	6386.737	64.43	40.0

Figure 11: Models Execution Time Summary

Received: July 08, 2025 ; Published: September 19, 2025

International Journal of Applied Mathematics
Volume 38 No. 2s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

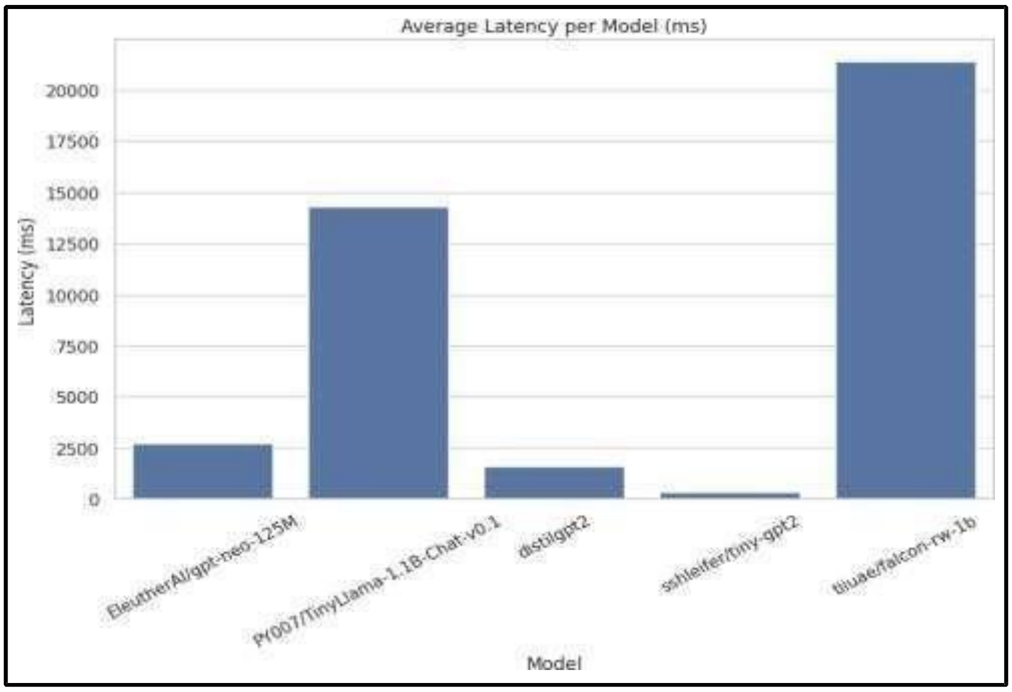


Figure 12: Average Latency per Model

5.3 Resource Utilisation Patterns

Similar to the results for latency, the models' memory usage confirmed that Falcon-RW-1B was the most memory-intensive model, requiring an average of 6386 MB of RAM. TinyLlama used almost 5242 MB, more than any other model, whereas Tiny-GPT2 only needed slightly more than 2 GB. CPU use was more consistent across the models, yet somewhat matched their complexity. Falcon and TinyLlama reached unusually high CPU loads, with scores of 62% and above, while DistilGPT2 and GPT-Neo had significantly lower loads at 53% and 59%, respectively. Although a few models excel in benchmarks, their extra processing time may be too much for Google Colab or similar devices. In general, resource profiles reveal that the new models are designed for excellent performance rather than efficient use, so it is necessary to analyse the costs and benefits before using them.

Received: July 08, 2025 ; Published: September 19, 2025

International Journal of Applied Mathematics
Volume 38 No. 2s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

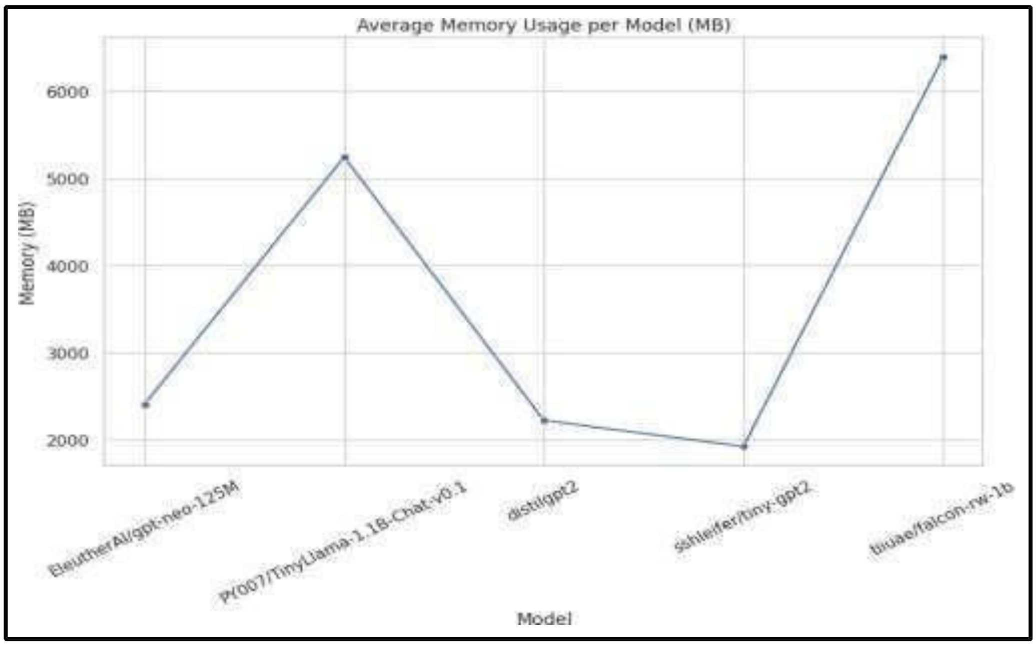


Figure 13: Resource Utilisation per Model

5.4 Performance Tradeoffs

It is clear from the scatterplot that a tradeoff exists when deploying LLMs. GPT-Neo set a new benchmark record (42.0) and performed well, achieving a low latency of 2.7 seconds. Still, Falcon-RW-1B performed similarly on the benchmarks (40.0), yet was almost 8 times slower. Although TinyLlama achieved a decent score (39.0), it took a considerable amount of time for the model to process each sample. As predicted, the benchmark scores for DistilGPT2 and Tiny-GPT2 were lower (31.0 and 25.0, respectively), and their inference was also faster. Such a tradeoff enables software developers to choose between speed and timely response. Because GPT-Neo and DistilGPT2 were near the performance-efficiency line, they are the best choices for low-cost operations that still yield excellent results in accuracy tests.

Received: July 08, 2025 ; Published: September 19, 2025

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

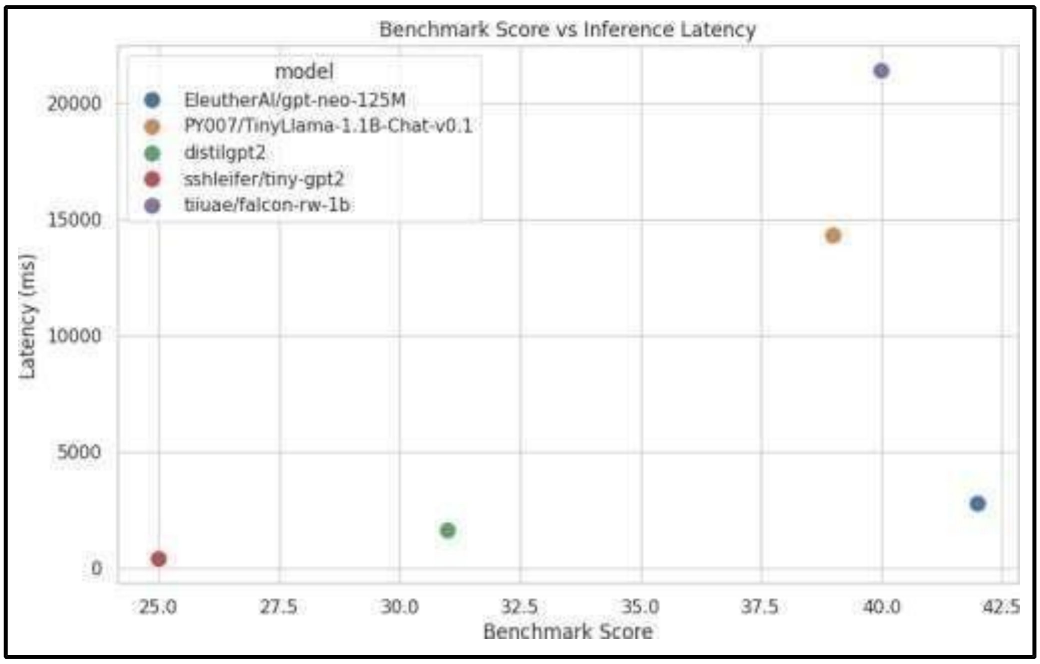


Figure 14: Benchmark Score vs Inference Latency

5.5 Latency Consistency

Using boxplots, changes in the amount of variance for latency are illustrated. With the fewest outliers, Tiny-GPT2 demonstrated that it is well-suited for consistent inference use. DistilGPT2 experienced some variations, although the outcomes were suitable for most large-scale companies. The distribution of GPT-Neo remained unchanged, despite being a larger model. Falcon and TinyLlama showed noticeable changes in their response speed. There were some apparent differences between TinyLlama and Falcon: TinyLlama yielded more consistent results, while Falcon's timing was less reliable, and fewer prompts were used. This means that there may be increased risks in production where guaranteeing a timeline is necessary. As a result, checking consistency gives us more information, suggesting that the average latency and stability should be evaluated.

Received: July 08, 2025 ; Published: September 19, 2025

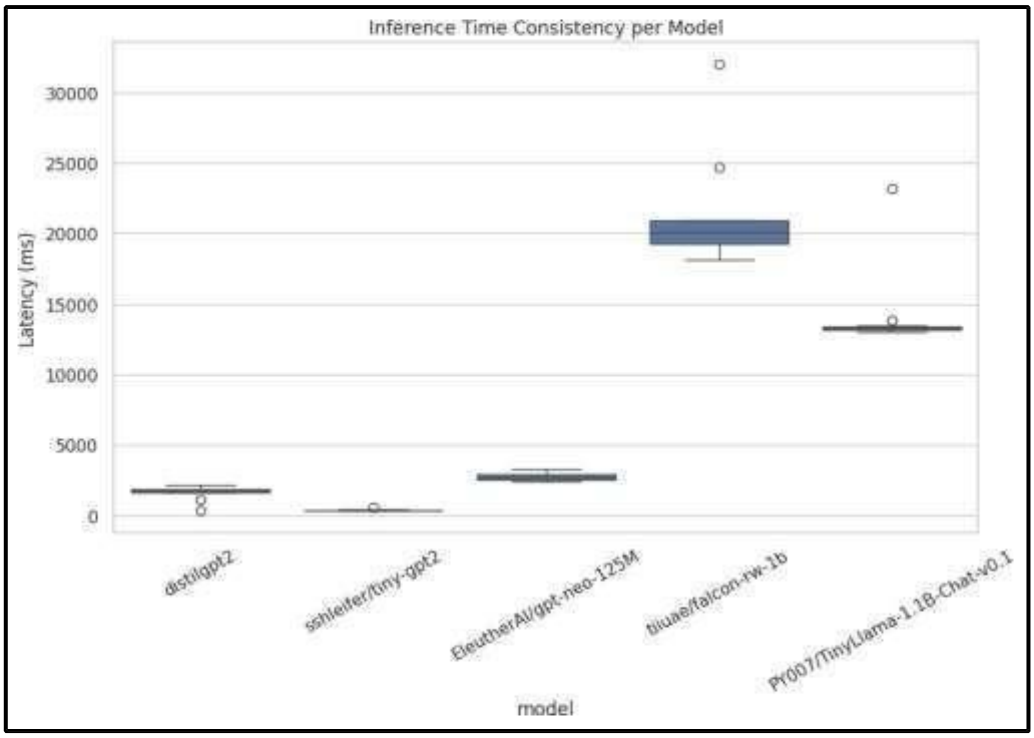


Figure 15: Inference Time Consistency per Models

5.6 Cost Estimation

Recording the expected runtime of an LLM model on a simulated dataset and comparing it to different pricing plans on the cloud helps determine if LLM can be used affordably. Since the cost is virtually nothing on the free CPU tier of Google Colab, it is perfect for exploring and studying. Even so, the inference results obtained using the Colab GPU at $0.75/hr and the GCP A100 at $0.40/hr differ. Our runtime for all models was around so many hours, resulting in a GPU cost of $0.083 and an A100 GCP cost of $0.044. Even though the initial numbers may be just a representation, they rapidly accumulate when tested on a larger scale. Significant infrastructure costs will be incurred if thousands of prompts are served in a single batch each day. The estimates reveal that inference should be optimised for both speed and cost, so lighter models are suitable wherever the necessary performance can be achieved. Despite achieving top benchmark results, Falcon and TinyLlama are more expensive due to their lengthy computation times.

Received: July 08, 2025 ; Published: September 19, 2025

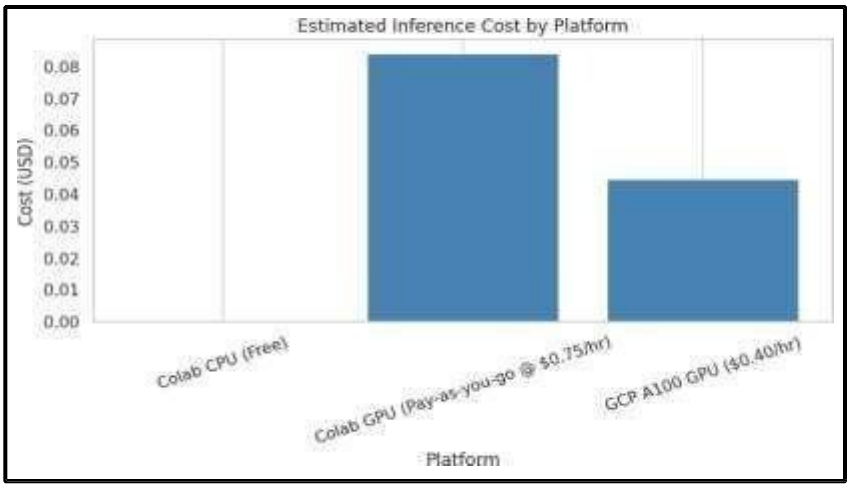
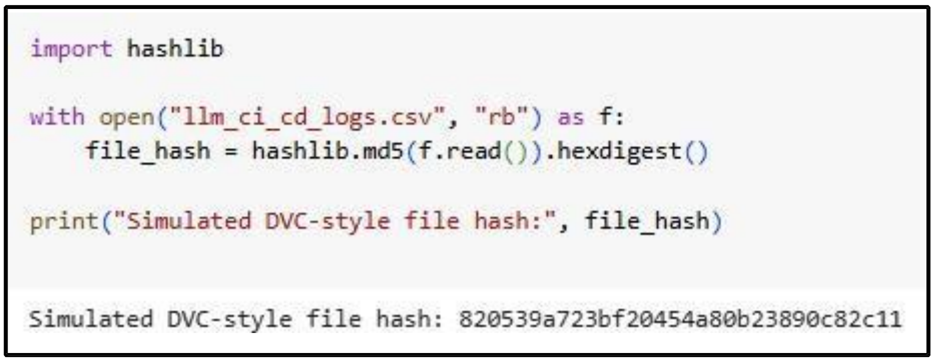


Figure 16: Cost Estimation

5.7 DVC-style Hash Simulation

DVC-style file tracking with hashlib.md5 was added to make models reproducible and facilitate version control. A unique fingerprint was created for every CSV log obtained from the CI/CD process. For example, the experiment's log file contains a hash, 820539a723bf20454a80b23890c82c11. You can use this identifier to ensure that results are consistent and identical during ML experiments, which is necessary for auditing purposes. Also, successfully initialised MLflow, registered it as the LLM_CI_CD_Benchmark experiment and logged latency and benchmark results for every model used. With these two steps, the simulation of CI/CD is complete, as it provides the starting point for versioning, rollback, and logging of what happened. Because of this, they are instrumental in MLOps applications that cannot fully utilise GitHub Actions and Docker orchestration due to limited system resources.

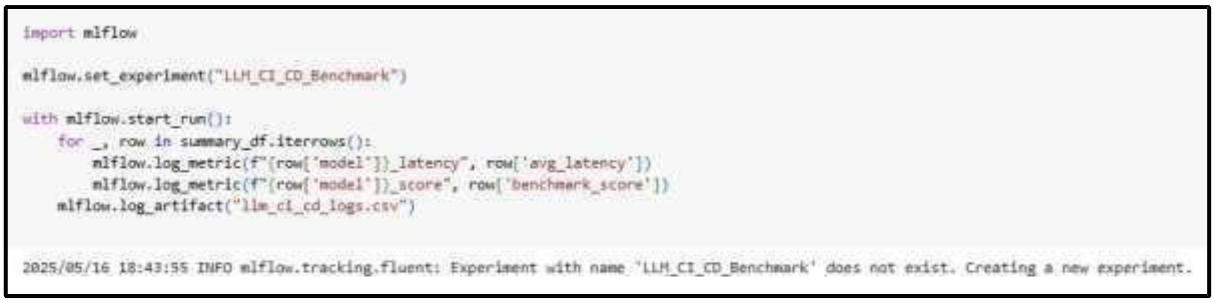


```
import hashlib

with open("llm_ci_cd_logs.csv", "rb") as f:
    file_hash = hashlib.md5(f.read()).hexdigest()

print("Simulated DVC-style file hash:", file_hash)
```

```
Simulated DVC-style file hash: 820539a723bf20454a80b23890c82c11
```

Figure 17: DVC-Style File Tracking

Received: July 08, 2025 ; Published: September 19, 2025



```
import mlflow

mlflow.set_experiment("LLM_CI_CD_Benchmark")

with mlflow.start_run():
    for _, row in summary_df.iterrows():
        mlflow.log_metric(f"{row['model']}_latency", row['avg_latency'])
        mlflow.log_metric(f"{row['model']}_score", row['benchmark_score'])
    mlflow.log_artifact("llm_ci_cd_logs.csv")

2025/05/16 18:43:55 INFO mlflow.tracking.fluent: Experiment with name 'LLM_CI_CD_Benchmark' does not exist. Creating a new experiment.
```

Figure 18: MLflow Integration

6. Discussion

Based on CI/CD pipelines, LLM evaluations help reduce the risk of strange results caused by different infrastructures. When software is deployed as a traditionally run program, changes in performance across various hardware and environments make it challenging to replicate and scale the work. However, using Google Colab to set up a CI/CD pipeline, this research finds that automation enables developers to distinguish model outcomes from the platform and achieve reliability through logging, validation, and version control. This type of abstraction is necessary as LLMs are used in production, as unexpected model failures sometimes result from environmental mismatches.

With MLflow and DVC-style hashing used in an LLM pipeline, reproducing the work becomes much easier. Thanks to assigning a version hash for each run and logging metrics in a trackable MLflow experiment, the framework enables developers to verify the exact conditions required to produce any given result. This becomes necessary when working together or in fields where it is vital to verify how models function. They also allow groups to avoid accidentally losing their progress by allowing a quick transition back to past approved states.

Easy benchmarking at the start of the model selection process provides valuable insights into the process. The framework provides information about costs and latency during the early stage, when we are still in the prototype stage. Deployment plans here are based on data and correspond with operational rules. This study reveals that, compared to less accurate models such as Tiny-GPT2, these models operate very quickly. In contrast, models like Falcon-RW-1 B and TinyLlama record higher scores but are significantly slower and consume more RAM.

Deployment teams need to measure the average performance of models and how much they vary with different amounts of prompts. We can determine how a model behaves under various types of stress by using latency standard deviation and box plots. Otherwise, a sizable memory footprint could prevent a model from running on mobile devices or simple edge servers. Overall, this framework helps to balance decisions by ensuring models are compatible with the infrastructure environment.

7. Conclusion and Future Work

This study demonstrates that CI/CD can facilitate automation, reproducibility, and performance monitoring of LLM tasks, particularly on platforms with limited resources. We

Received: July 08, 2025 ; Published: September 19, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

built the framework by using Google Colab, Hugging Face Transformers, MLflow, and lightweight hashing mechanisms. The evaluation of five open-access LLMs demonstrated that this framework effectively monitors key metrics and ensures the study's reproducibility. It does so in a way that maintains easy access to the LLM source code.

We have made three main contributions. First, we demonstrated how to utilise the Open LLM Benchmark dataset with a modular CI/CD pipeline, offering a method for testing in any environment. Furthermore, this project utilises reproducibility tracking with Colab, incorporating MLflow and DVC-style hash simulations, to address a common issue in the literature: cloud-based benchmarking lacks reliable versioning. Next, we developed a set of resources to visualise how the system operates, providing model deployment teams with valuable insights for managing the system's speed and cost.

Nevertheless, some problems still need to be addressed. The CI/CD pipeline was simulated in Colab notebooks and was not implemented on Docker, Jenkins, or Kubernetes. Notably, the model was not trained, and its inference performance was only evaluated. The findings are not ideal for production, yet the principles and procedures can be used in many fields.

At this point, the framework should be expanded to incorporate real-world use cases. Implementing FastAPI for serving enables the benchmarking process to conclude with an API interface for services such as GCP Vertex AI. Another approach is to add an MLflow registry, allowing workplace models to be deployed and monitored repeatedly in production. The system will operate more realistically by using Streamlit for live dashboards and testing the first request time. With these advancements, the usage of CI/CD will make the final deployment of LLMs more effective and straightforward.

References

[1] Myers, D., et al., Foundation and large language models: fundamentals, challenges, opportunities, and social impacts. Cluster Computing, 2024. **27**(1): p. 1-26.

[2] Chkirbene, Z., et al. Large Language Models (LLM) in Industry: A Survey of Applications, Challenges, and Trends. in 2024 IEEE 21st International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT (HONET). 2024. IEEE.

[3] Prasad, V.K., et al., Efficient resource utilization in IoT and cloud computing. Information, 2023. **14**(11): p. 619.

[4] Alaharju, H., Ensuring Performance and Reliability in LLM-Based Applications: A Case Study. 2024.

[5] Hadi, M.U., et al., Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. Authorea Preprints, 2023. **1**: p. 1-26.

Received: July 08, 2025 ; Published: September 19, 2025

[6] Shao, M., et al., Survey of different large language model architectures: Trends, benchmarks, and challenges. IEEE Access, 2024.

[7] Shoeybi, M., et al., Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053, 2019.

[8] Wang, P., et al., Cost-effective and latency-minimized data placement strategy for spatial crowdsourcing in multi-cloud environment. IEEE Transactions on Cloud Computing, 2021. **11**(1): p. 868-878.

[9] Bai, G., et al., Beyond efficiency: A systematic survey of resource-efficient large language models. arXiv preprint arXiv:2401.00625, 2024.

[10] Mathur, P., Cloud computing infrastructure, platforms, and software for scientific research. High Performance Computing in Biomimetics: Modeling, Architecture and Applications, 2024: p. 89-127.

[11] Lee, W.-K., et al., High throughput lattice-based signatures on gpus: Comparing falcon and mitaka. IEEE Transactions on Parallel and Distributed Systems, 2024. **35**(4): p. 675-692.

[12] McMahon, A.P., Machine Learning Engineering with Python: Manage the lifecycle of machine learning models using MLOps with practical examples. 2023: Packt Publishing Ltd.

[13] Kolawole, I. and A. Fakokunde, Improving Software Development with Continuous Integration and Deployment for Agile DevOps in Engineering Practices.

[14] Ugwueze, V.U. and J.N. Chukwunweike, Continuous integration and deployment strategies for streamlined DevOps in software engineering and application delivery. Int J Comput Appl Technol Res, 2024. **14**(1): p. 1-24.

[15] Suddala, S., Automating the Data Science Lifecycle: CI/CD for Machine Learning Deployment.

[16] Heller, P., Automating Workflows with GitHub Actions: Automate software development workflows and seamlessly deploy your applications using GitHub Actions. 2021: Packt Publishing Ltd.

[17] Mishra, P., Providing Practical Guidance, in A Guide to Implementing MLOps: From Data to Operations. 2025, Springer. p. 9-86.

[18] Raj, E., Engineering MLOps: Rapidly build, test, and manage production-ready machine learning life cycles at scale. 2021: Packt Publishing Ltd.

[19] Baine-Omugisha, M., Automated Program Repair Through Natural Language Processing in a DevOps Pipeline System. 2024.

[20] Topsakal, O., Evaluating the Performance of Large Language Models (LLMs) Through Grid-Based Game Competitions: An Extensible Benchmark and Leaderboard on the Path

Received: July 08, 2025 ; Published: September 19, 2025

to Artificial General Intelligence (AGI). The Journal of Cognitive Systems, 2024. **9**(2): p. 8-19.

[21] Myrzakhan, A., S.M. Bsharat, and Z. Shen, Open-llm-leaderboard: From multi-choice to open-style questions for llms evaluation, benchmark, and arena. arXiv preprint arXiv:2406.07545, 2024.

[22] Zheng, S., An systematic evaluation on leading large language models and their factuality investigation as question answering systems. 2024.

[23] Babar, Z., A study of business process automation with DevOps: A data-driven approach to agile technical support. American Journal of Advanced Technology and Engineering Solutions, 2024. **4**(04): p. 01-32.

[24] Raiaan, M.A.K., et al., A review on large language models: Architectures, applications, taxonomies, open issues and challenges. IEEE access, 2024. **12**: p. 26839-26874.

[25] Liang, C., ON PARAMETER EFFICIENCY OF DEEP NEURAL LANGUAGE MODELS. 2023.

[26] Lamaakal, I., et al., Tiny language models for automation and control: Overview, potential applications, and future research directions. Sensors, 2025. **25**(5): p. 1318.

[27] Raghul. Open LLM performance benchmark. 2023; Available from: https://www.kaggle.com/datasets/itsraghul/open-llm-performance-dataset.

Received: July 08, 2025 ; Published: September 19, 2025