Journal of Insights in Basic and Applied Sciences
مجلة رؤى في العلوم الأساسية والتطبيقية
ISSN: 3106-1737
Volume 2, Issue 4, 2026, Pages: 462-473
Journal homepage: https://jibas.com.ly/index.php/jibas/en

Performance Evaluation of Single-Instance and Replicated Deployment of a CPU-Based Libyan Arabic Sentiment Analysis Service

Souad Alharm *
Department of Internet technologies, Information Technology, Al-Asmariya Islamic University, Zliten, Libya

تقييم أداء النشر بنسخة واحدة والنشر بنسخ متعددة لخدمة تحليل المشاعر بالعربية الليبية على المعالج المركزي

سعاد الهرم *
قسم تقنيات إنترنت، تقنية المعلومات، الأسمرية الإسلامية، زليتن، ليبيا

*Corresponding author: s.alharm@it.asmarya.edu.ly

Received: September 10, 2026	Accepted: October 15,2026	Published: November 20, 2026

Copyright: © 2026 by the authors. This article is an open-access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

Abstract:
Despite the importance of classification Accuracy in evaluating natural language processing models, their operational performance under concurrent workloads emerges as a critical factor when running on a CPU only. This study investigates the real-world performance of a fin-tuned MARBERT-based Libyan dialect sentiment analysis service in terms of throughput, response latency, resource utilization consumption, and failure rates across multiple deployment and load-balancing configurations.
In model evaluation on 423 labeled comments using Three-seed five-fold cross-validation. AraBERT, MARBERT and CAMelBERT achieved average accuracies of (89.2%, 88.3%, and 85.9%; respectively , with on decisive statistical difference between AraBERT and MARBERT). When deploying the fine-tuned MARBERT. Via FastAPI[26] on a CPU only within a dual-core container, a six-core container or three dual-core replicas behind Nginx[27], performance was roughly equal at 10 concurrent users at approximately 4.6 requests/second for the dual-core one; however, this advantage did not remain consistent in subsequent sessions, as the two larger configurations converged at 100 users around 22 requests/second and absolute throughput varied by up to twofold across sessions while single-container modifications 6workers or 3 workers with 2 threads did not exceed about 13 requests/second. Under Round Robin distribution at 100 users, the average response time of the slowest replica reached between 6 and 35 times that of the fastest, while Little's Law matched user counts within an error margin not exceeding 6%.
In additional tests at 100 concurrent users, the least connections policy clearly improved load balancing reducing the response time ratio between the slowest and fastest replicas from roughly 38 to nearly 1.2, while increasing throughput by about 10% and cutting 95th percentile p95 response latency from 11.0 to 4.9s . Nevertheless, these results remain preliminary supporting indicators rather then conclusive evidence due to limiting experiments to three replications on a single machine.

Keywords: Arabic transformer models; Libyan dialect; sentiment analysis; CPU-only inference; vertical scaling; horizontal replication; load balancing; tail latency; load testing; performance evaluation.

الملخص
بينما تعتبر دقة التصنيف امرا أساسيا لتقييم نماذج معالجة اللغة الطبيعية، فالأداء الفعلي تحت احمال عمل متزامنة وواقعية لا يقل عنها أهمية ، ولا سيما حين تعتمد البنية التحتية على المعالج المركزي فقط. في هذه الدراسة قيم الأداء الفعلي لخدمة تحليل المشاعر باللهجة الليبية العربية المبنية على MARBERT بعد تحسينها ، مع التركيز على معدل الإنجاز ، ووقت الاستجابة ، واستخدام الموارد ومعدلات الفشل عبر تكوينات نشر متعددة وسياسات موازنة احمال مختلفة.
خضت ثلاث نماذج للتدريب على 423 تعليقا موسوما، وهي CAMeLBERT , MARBERT, AraBERT باستخدام تحقق متقاطع خماسي تكرر بثلاث بذور بلغت متوسطات الدقة على الترتيب 89.2%, 88.3%,85.9% دون وجود فروق ذي دلالة إحصائية بين AraBERT , MARBERT وقد جرى تقديم نقطة تفتيش MARBERT المضبوطة عبر FastAPI و Nginxعلى المعالج المركزي حصرا، في ثلاثة تكوينات : حاوية واحدة بمعالجين ، وحاوية واحدة بستة معالجات، وعدد ثلاث نسخ بمعالجين لكل منها خلف Nginx.
تقاربت الإنتاجية في التكوينات عند 10 مستخدمين متزامنين، تقاربت الإنتاجية في التكوينات كافة عند نحو 4.6 طلب في الثانية، وفي الحملة الرئيسية كان النشر المتكرر الأكثر إنتاجية بدءا من 50 مستخدما اذ تراوحت الإنتاجية بين 15 -17 طلباً في الثانية، مقارنةً بنحو 11 طلباً في الثانية للحاوية ذات ست معالجات، ونحو 5 طلبات في الثانية للحاوية ذات معالجين. غير أن هذا الترتيب لم يستقر في جلسات لاحقة، وتفاوتت الإنتاجية المطلقة بين الجلسات بما يصل إلى الضعف. وفي في ظل التوزيع الدوري.(Round Robin) بلغ متوسط زمن استجابة أبطأ نسخة من 6 إلى 35 ضعف متوسط أسرع نسخة، وساهمت سياسة الاتصالات الأقل(Least Connection) في تحسين توازن الحمل بين النسخ بصورة واضحة، إذ انخفضت نسبة زمن الاستجابة بين أبطأ نسخة وأسرعها من نحو 38 إلى نحو 1.2 ، بالتزامن مع زيادة الإنتاجية بنحو 10% وانخفاض زمن الاستجابة عند المئين الخامس والتسعين (P95) من 11.0 ثانية إلى 4.9 ثانية. ومع ذلك، ينبغي التعامل مع هذه النتائج بوصفها مؤشرات داعمة لا أدلة حاسمة، نظراً إلى أنها استندت إلى ثلاث تكرارات فقط نُفِّذت على جهاز واحد.

الكلمات المفتاحية: المحولات العربية؛ اللهجة الليبية؛ تحليل المشاعر؛ الاستدلال على المعالج المركزي؛ التوسع الرأسي؛ التكرار الأفقي؛ موازنة الحمل؛ زمن الاستجابة الطرفي ؛ اختبار الحمل؛ تقييم الأداء.

1. Introduction

With the increasing reliance on transformer language models. Transformers in processing Arabic texts the success of these models is no longer limited to their superiority in theoretical accuracy test but has extended to include their operational efficiency. when deployed as live software services responding to real-time user requests.
The problem centers on the limitation of evaluating Arabic NLP models solely by classification accuracy whereas. The practical effectiveness of sentiment classifiers such as analyzing Libyan dialect comment via HTTP interface requires speed and efficiency under concurrent request pressure the import a CPU stands out in resource constrained research environments that lack graphics processing units GPUs and are limited to a single server as behavior of these models differs from traditional web because each inference. A CPU core for tens or hundreds of milliseconds creating intense competition for limited computing resources.
AS for the research gap it lies in the scarcity of empirical studies examining the operational behavior of low resource Arabic dialect services post training particularly in the TRADE-OFF between vertical scaling allocating more resources to a single container and horizontal replication running multiple replicas behind a reverse proxy within a fixed CPU budget. Although both options are widely discussed in the literature on conventional web applications [11],To represent this case practically MARBERT was selected as a pretrained language model well suited.

For processing Arabic dialects noting that the generalizability of deployment results to other transformer services depends on model architecture sequence length and tokenization behavior.

This paper measure the performance of a Libyan dialect sentiment analysis service on a CPU-only machine keeping the model and request data constant. While changing only the deployment architecture.

1. How does vertical scaling a single larger container compare to horizontal replication. Identical replicas behind an Enginx proxy in terms of throughput tail latency CPU utilization and failure rates as concurrent user increase?
2. How does the load balancing policy effect reducing performance disparity among replicated inter.

The contributions are :

- Deployment Configuration Comparison: A controlled comparison of three CPU-budgeted under 10 to 150 concurrent users with three replications per condition, measuring throughput response latency percentiles (including p95 and p99), failure rates and timeouts.
- Resource Analysis and Load Balancing : (CPU utilization and per-request CPU cost), verified for consistency via Little's LAW.
- alongside diagnosing round-robin from proxy logs and evaluating the least-connections policy at 100 users.

- Operational Configuration Tuning: Highlighting the impact of practical settings (such as thread counts within CPU quotas and model warm-up) in shaping performance outcomes
- A validation of the workload through repeated cross-validation of three Arabic models on 423 labeled comments.

2. Background and Related Work

The related work is reviewed across four themes:

2.1 Arabic Pretrained Models and Sentiment Analysis

Arabic pretrained models have advanced sentiment analysis: AraBERT [1] adapts BERT [23] to Arabic, MARBERT as a model trained on tweets converting multiple Arabic dialects [2], and CAMeLBERT studies the effect of language variety and Modern Standard Arabic versus mixed dialects[3]. For the Libyan dialect, MARBERTv2 and AraBERTv2 reached 71% and 70% on a small set of comments [4]; Multinomial Naïve Bayes with TF–IDF reached achieved 84.4% on 4,529 restaurant comments [5], and Decision Tree outperformed SVM and Naïve Bayes, achieving 72% in another studay [6]. These studies evaluate classification accuracy but do not examine model-serving costs or behavior under concurrent workloads.

2.2 Transformer Inference Serving and Optimization.

On the serving side, Clipper introduced batching and cashing for online prediction[7], and Wang et al , studied scheduling of Transformer inference on GPU servers [8]. On CPUs, dynamic INT8 quantization accelerated BERT-base and DistilBERT by factors of 2.29–2.33 with accuracy degradation of a most 5.1% [9], and optimizing the most expensive components of the Transformer gave up to 2.37×speedup [10]. Evidence on Arabic transformer models in CPU-only containers under constrained resource budget remains limited.

2.3 Single-instance versus replicated deployment, and load balancing

Replication and load balancing are mostly studied for microservices: Villamizar et al. compared monolithic and microservices deployment [11], Dragoni et al. reviewed their challenges [12]. Gan et al. introduced Death Star Bench [13], and Camilli et al. studied performance models [14].The balancing policy matters: behind behind Nginx in Docker [25], Round Robin had the highest p99 latency at high load and , and its p99 rose by 827%, when hen half the capacity was lost, whereas Least Connection and Weighted Round Robin stayed stable[15]. Least Connections–based load balancing in a reverse-proxy deployment [16]. Pais et al, identified load balancing, fault tolerance, and scalability as essential for cloud NLP services [17] . and containers and orchestration make horizontal scaling central to performance [18].Ganiev et al. served a fine-tuned BERT emotion classifier as a gRPC service on Kubernetes, called in batches by an Apache Spark job [19] .After tunning the serving runtime and increasing the request pressure from the client, they raised throughput from about 3.3 to about 300 requests per second with the same batch-job resources. They also reported that thread settings of the serving runtime affected throughput, and that a Layer-7 load balancer was needed to spread requests across replicas because gRPC multiplexes requests over persistent connections.

2.4 Performance Modeling and Operational Analysis.

Operational laws provide a framework for performance analysis based on measurable quantities rather than distributional assumptions. Little's Law relates the number of requests in a system to the arrival rate and the time spent in the system [20]. Denning and Buzen [21] developed operational analysis for queueing networks. These methods differ from the M/M/C model, which assumes Poisson arrivals and exponentially distributed service times. Such assumptions may not suit workloads involving concurrent users with think times, for which a closed workload model is more appropriate. Jia et al. used Little's Law to evaluate different scheduling policies in data centers and large-scale systems, and proposed estimators based on the law to reduce bias and improve the evaluation of scheduling policies [22].

2.5 Research Gap

TO the best of our knowledge, research examining the effects of deployment alternatives and load-balancing policies on Arabic transformer-based sentiment classifiers remains limited. This gap motivates empirical

comparisons of the two deployment approaches under comparable workloads, which is the aim of the present study.

3. Material and methods

3.1 Dataset

The dataset comprises 499 short comments in Libyan Arabic collected from social media about the Your Salry Instantly system in July 2025. Of these, 423 were carry binary sentiment labels (198 labeled 0, positive; 225 labeled 1, negative),and 76 were unlabeled and were excluded from training and evaluation. A majority class classifier that always predicts the (negative) reaches 53.2% an accuracy. The comments are short (mean: 6.8 words; median: 6; range: 2–26). Labels were irrelevant to the load-testing, so all 499 comments served as request payloads.

3.2 Models and Classification Experiment

Three models based on the BERT-base architecture [23] were compared: AraBERT [1] (aubmindlab/bert-base-arabertv02), MARBERT [2] (UBC-NLP/MARBERT), and CAMeLBERT [3] (CAMeL-Lab/bert-base-arabic-camelbert-mix). was fine-tuned with a learning rate of 2e-5, a batch size of 4, and 8 epochs, without warm-up or weight decay , using mixed precision on an RTX 4050 GPU; the model after the last epoch was used, and no tuning or checkpoint selection was performed on the evaluation folds. Evaluated used repeated five-fold stratified cross-validation with seeds (11, 22, and 33), for a total of 45 training runs. Accuracy and macro-F1 were pooled over all 423 comments and averaged over seeds, with 95% bootstrap intervals (10,000 resamples) and with exact McNemar tests on paired predictions, computed separately for each seed, with Holm correction across the three model pairs within each seed. and with exact McNemar tests on paired predictions, computed separately for each seed, with Holm correction across the three model pairs within each seed. The checkpoint used for serving (Section 3.3) was fine-tuned earlier with the same hyperparameters, but in FP32 with random seed 42. on 338 comments (a stratified 80% split, random_state = 42), and the model after the final step (680) was deployed, no evaluation during training. On the remaining 85 held-out comments it reached 91.8% accuracy (95% Wilson interval 84.0–96.0%) and a macro-F1 of 0.917; the cross-validation estimate for MARBERT (88.3%).

3.3 Serving Application and Deployment Configurations

The serving application uses FastAPI [26], run with Uvicorn. A POST request to the /predict endpoint contains a single comment. The server tokenizes the input (up to a maximum of 128 tokens), passes it to the model without computing gradients, selects the highest-scoring class, and returns the predicted label together with the measured inference time. The endpoint is synchronous, so concurrent requests are executed in a thread pool within each server process.The CPU-only build of PyTorch 2.2.2 was used, GPU access was disabled, and tokenizer files were loaded from a local cache, so no network access was required at runtime. All systems used images built from the same Dockerfile. SHA-256 checks confirmed that the model weights were byte-for-byte identical across all images used. Each server process was limited to one computational thread (OMP_NUM_THREADS and MKL_NUM_THREADS were both set to 1); the rationale is explained in Section 3.6.Three primary configurations were compared (Table 1), with further configurations in (Section 4.6 and 4.7). Mono-3x and Micro-3 share the same (6CPUs and 6 GB) service budget; Micro-3 additionally uses one CPU and 256 MB for Nginx, so service budgets are equal, whereas the total resources differ. Mono-1x is a reference with one third of this budget. CPU was limited with Docker CPU quotas, and all containers and the load generator ran on one host (Intel Core i5-13420H, 16 GB RAM)

Table 1. Deployment configurations and their resource budgets. The totals include the Nginx proxy where one is used.

#	Configuration	Deployment	Scaling type	CPUs(service + proxy)	Memory (service + proxy)	Total budget	Request distribution
1	Mono-1x	One container, one process	Reference	2+0	2 GB+0	2 CPUs,2 GB	Direct connection
2	Mono-3x	One container, one process	vertical	6+0	6 GB+0	6 CPUs,6 GB	Direct connection

3	Micro-3	Three identical replicas behind Nginx	Horizontal	6 (3 x 2) + 1	6 GB (3x2GB)+256MB	7 CPUs, 6.25 GB	Round robin
4	Micro-3-LC (Section 4.6)	Three identical replicas behind Nginx	Horizontal	Same as Micro-3	Same as Micro-3	7 CPUs, 6.25 GB	Least connections (least_conn)
5	Mono-3W (Section 4.6)	One container, three Uvicorn processes	Vertical	6+0	6 GB+0	6 CPUs, 6 GB	Direct connection

3.4 Load-Testing Protocol

Workload traffic was synthesized via Locust 2.47[28] . Each simulated user transmitted a randomly selected review then pauses for one to three seconds (uniform distribution, averaging two seconds), prior to dispatching subsequent requests thereby adhering to a closed workload paradigm. Four concurrency tiers were evaluated: 10, 50, 100, and 150 virtual clients. Users count escalated incrementally at one-tenth of the target rate per second, with each lasting 300 seconds. Request timeout was set at 30-second, and queries exceeding this threshold were classified as failures. Every condition underwent three replications. To mitigate gradual hardware performance drift, architectures were interleaved across each user tier iteration by iteration, rather than finishing all runs of one system before advancing to the next. Before each test, all containers were purged and solely the target system was launched. After verifying that all server processes were ready, 200 warm-up requests using randomly chosen input with a fixed seed were dispatched by a five-second pause, the actual benchmark and then a 20-second cooldown interval

3.5 Metrics and Validation Checks

For each run, Locust provided throughput, mean, median, p95and p99 response times, and failures; docker stats (sampled about every 2s). provided CPU and memory use; for Micro-3, Nginx logs gave the proxy time and replica time of each request. Values are mean of three repetitions with 95% t-based confidence intervals (two degrees of freedom, hence wide); percentiles are approximate because Locust bins response times. Each run was checked with Little's Law by comparing N, with X(R + Z), where X is throughput, R is mean response time, and Z is mean think time (2s). CPU cost per request is mean number of busy cores by divided by throughput.

3.6 Preliminary Findings That Changed the Experimental Configuration

Preliminary experiments changed the configuration in three ways. First, an isolated request took 0.048s with two threads and 0.065s with one thread in a two-CPU container. but 0.195s with four threads because of were by the CPU quota throttling. Second, in a 60-second test with 10 users, Mono-1x, configured with two threads per process, completed 133 requests (median: 1.4 seconds; p95: 5.1 seconds) at about 200% CPU, while with one thread per process, it completed 250 requests (median: 0.14 seconds; p95: 0.35 seconds), at 50% -60%. CPU; all systems were use one thread per processThird, the first request to each replica took between 1 and 2.5 seconds. Warming the system with a single repeated sentence did not eliminate this effect; consequently, the warm-up procedure used a diverse set of real comments.

4. Results and discussion

4.1 Classification Performance

Mean pooled accuracy was 89.20% for AraBERT, 88.34% for MARBERT, and 85.89% for CAMeLBERT. Against a 53.19% majority-class baseline(Table 2) , between AraBERT and MARBERT was 0.87 percentage points 95% interval −1.18 to +2.92), so these two models cannot be distinguished. CAMeLBERT was 3.3 points below AraBERT and 2.4 points below MARBERT, with bootstrap intervals of these differences excluded zero. However, per-seed exact McNemar tests with Holm correction reached significance in only one of the three seeds (adjusted $p = 0.035$; all others ≥ 0.196), so this gap is suggestive rather than conclusive. Seed variation was small (standard deviation: 0.14–0.90 percentage points), and the main source of uncertainty is the limited number of labeled comments ($n = 423$), which makes all intervals wide.MARBERT was used in the deployment study because it targets dialectal and social-media Arabic, and a single model keeps differences attributable to deployment rather to the language model.

Table 2. Classification performance (repeated five-fold stratified cross-validation with three random seeds, n = 423 comments; 95% intervals by bootstrap over comments; SD over the three seeds). The majority class is the negative class.

#	Model	Accuracy (%)	95% confidence interval	Macro-F1	SD of accuracy over seeds (points)
1	AraBERT	89.20	[86.45, 91.80]	0.891	0.14
2	MARBERT	88.34	[85.50, 91.02]	0.883	0.90
3	CAMeLBERT	85.89	[82.74, 88.81]	0.858	0.49
4	Majority class	53.19	Not applicable	Not applicable	Not applicable

4.2 Load-Testing Results

Table 3. p95 and p99 are the mean of per-run values obtained from Locust's binned response times

#	Users	Deployment configuration	Throughput (requests/s)	Median (s)	p95 (s)	p99 (s)	Failures per run
1	10	Mono-1x	4.6	0.12	0.32	0.63	0
2	10	Mono-3x	4.6	0.12	0.25	0.37	0
3	10	Micro-3	4.6	0.12	0.22	0.26	0
4	50	Mono-1x	5.2	7.6	9.3	9.7	0
5	50	Mono-3x	11.3	2.5	3.7	4.1	0
6	50	Micro-3	17.0	0.27	3.0	3.3	0
7	100	Mono-1x	5.2	17.3	19.3	19.7	0
8	100	Mono-3x	10.7	7.4	8.7	9.5	0
9	100	Micro-3	15.3	1.5	13.7	14.0	1.3
10	150	Mono-1x	5.2	27.3	28.0	28.3	564
11	150	Mono-3x	10.6	12.5	14.0	14.3	0
12	150	Micro-3	15.5	8.1	17.7	20.3	1.3

At 10 users, all three systems processed approximately 4.6 requests per second, matching the workload generated by ten users with a cycle time of about 2.1 seconds. Thus, none of the systems was under significant pressure. The systems began to diverge at 50 users. Mono-1x stabilized at approximately 5.2 requests per second, regardless of the number of users. Mono-3x achieved 10.6–11.3, about twice Mono-1x, which had one third of its CPU allocation. and Micro-3 reached 17.0 at 50 users, then achieved 15.3–15.5 at 100 and 150 users, in the main campaign, the throughput ranking Micro-3> Mono-3x > Mono-1x held in all nine paired repetitions (Figure 1).

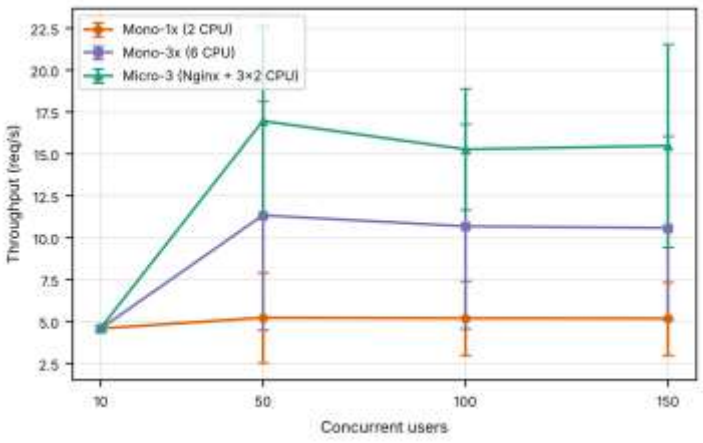


Figure 1. Throughput versus number of users (mean ± 95% confidence interval; three repetitions).

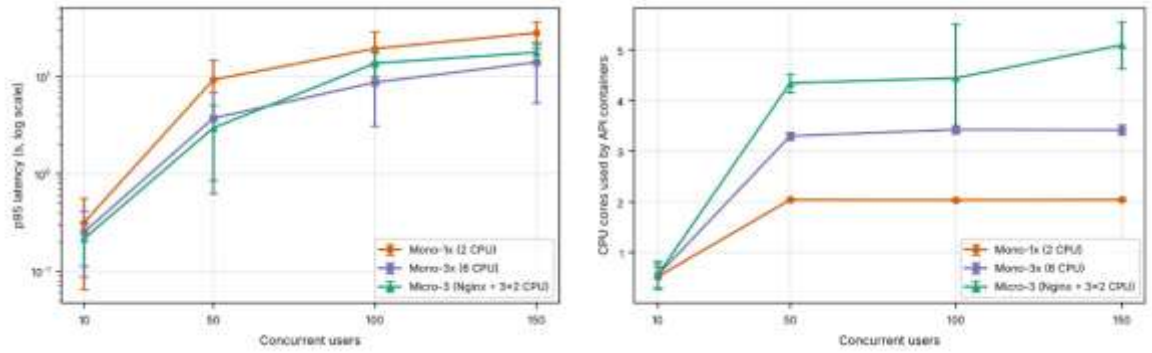


Figure 2. (a)p95 response time versus number of users (logarithmic scale; mean ± 95% confidence interval). (b) busy CPU cores in service containers versus number of users.

Because n = 3, the confidence intervals around individual means overlap (for example, 17.0 ± 5.7 requests/s for Micro-3 at 50 users). The results should therefore be interpreted in terms of ranking, rather than as precise speedup factors. The repetitions were not entirely independent: in several high-load conditions, the third and final run was approximately 15–45% faster than the first two. The cause is unknown and may have involved temperature or background activity on the device.

Tail latencies tell a different story from throughput. Micro-3 had substantially lower median response times at 100 and 150 users (1.5 and 8.1 seconds, respectively) than Mono-3x (7.4 and 12.5 seconds). However, its p95 was higher (13.7 and 17.7 seconds, compared with 8.7 and 14.0 seconds). This indicates a split in Micro-3 response times, with most requests completing quickly and a minority taking much longer. Section 4.4 investigates the cause. Near saturation, waiting dominates response time, consistent with the response-time equation for closed systems. For Mono-1x at 150 users, the equation $N/X - Z$ gives $150/5.2 - 2 \approx 27$ seconds, close to the measured mean of approximately 26 seconds.

4.3 Resource Utilization and Operational Analysis

Little's Law provided a good fit. The estimated mean number of users, $X(R + Z)$, ranged from 9.8 to 9.9, 48.7 to 49.1, 96.0 to 97.5, and 141.3 to 145.1 at 10, 50, 100, and 150 users, respectively. In all cases, these estimates were within 6% of the actual values. The largest deviation was associated with failures at 150 users for Mono-1x.Memory usage for each service container remained between 0.84 and 0.92 GiB, well below the 2 GiB limit, while Nginx used approximately 0.02 CPU cores. CPU utilization differed markedly across the systems: Mono-1x fully utilized its two CPU cores and was therefore CPU-bound. Mono-3x used only about 3.4 of its six available cores, yet was saturated; Micro-3 used between 4.3 and 5.1 of its six cores. CPU cost per request increased with load, from approximately 0.12 seconds at 10 users to 0.26–0.40 seconds—several times the 0.065 seconds measured for a single isolated request. This increase is partly attributable to processing overhead around the model and partly to contention among concurrent requests within the container.

4.4 Replica Imbalance in the Multi-Replica Deployment

Nginx logs show that round-robin scheduling distributed requests as expected: each replica received 32–35% of requests at 100 and 150 users. Response times, however, were highly uneven. At 10 users, the mean response time of the slowest replica was within 4% of that of the fastest. At 100 users, the ratio of the slowest to fastest replica ranged from 6.3 to 35 (median: 20); at 150 users, it ranged from 5.6 to 21 (median: 7.4). In one run, one replica's mean response time was 12.5 seconds, while those of the other two were 0.6 and 1.0 seconds. The slow replica varied between runs, indicating that the imbalance was not caused by a defect in a particular container. CPU measurements support this finding: at 50 users and above, one replica was typically near its limit (200%), while the other two operated at 100–130% (Figure 3). The most likely explanation is a feedback loop: round-robin scheduling does not account for load, so concurrent requests accumulate at a replica that is already delayed. Each request then consumes more CPU time when many threads compete for two cores, further increasing the delay while the other replicas remain underutilized. This also explains the gap between Micro-3's low median and high p95. The proxy added no measurable delay: the difference between total request time and replica time was approximately 10 ms or less. Section 4.6 directly tests this explanation.

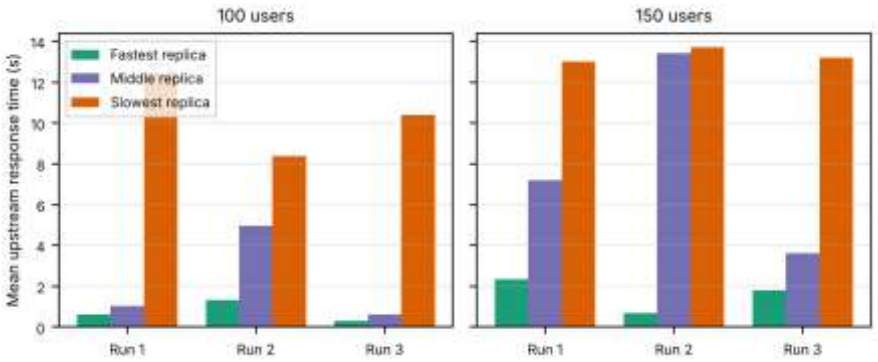


Figure 3. Mean response time for each Micro-3 replica in each run; replicas are ordered from fastest to slowest.

4.5 Failures

The main failures were client-side timeouts in Mono-1x at 150 users: 1,005 and 687 requests timed out in two of the three repetitions, while no timeouts occurred in the third. Micro-3 recorded a small number of HTTP 502 responses—8 out of 31,779 logged requests (approximately 0.03%)—possibly due to the use of stale keep-alive connections. Micro-3 also recorded 516 requests with status code 499. These were concentrated in the final second of each run and represent requests aborted when the load generator stopped; they are therefore not considered errors.

4.6 Supplementary Experiments at 100 Users

Two additional configurations were tested to investigate why Mono-3x lagged behind Micro-3. The two original configurations were also rerun during the same session.The first, Micro-3-LC, was identical to Micro-3 but used Nginx's least-connections policy (least_conn) instead oround-robin scheduling. The second, Mono-3W, used a single container with the same resource allocation—six CPUs and 6 GB of memory—but ran three Uvicorn processes instead of one.

Table 4. Supplementary experiments at 100 users (mean of three repetitions).

#	Configuration	Throughput (requests/s)	Median (s)	p95 (s)	p99 (s)	Busy CPU cores (out of 6)	CPU-seconds per request
1	Micro-3 (round robin)	16.8	1.6	11.0	11.7	4.6	0.27
2	Micro-3-LC (least connections)	18.5	3.2	4.9	7.0	5.9	0.32
3	Mono-3W (three processes)	14.0	4.9	7.0	7.6	6.1	0.43
4	Mono-3x (one process)	12.8	5.8	6.5	6.8	3.4	0.27

Least-connections policy. This alteration almost eliminated replica imbalance. The ratio of the slowest replica's mean response time to that of the fastest, which ranged from 9.9 to 41.9 under round-robin scheduling (median: 38), fell to 1.2–1.3. The proportion of requests with total response times exceeding five seconds decreased from 36% to 4%. Throughput increased from 16.8 to 18.5 requests/s (approximately 10%), p95 fell from 11.0 to 4.9 seconds, and p99 fell from 11.7 to 7.0 seconds. CPU utilization increased from 4.6 to 5.9 of the six available cores. The same pattern appeared in all three repetitions, although the throughput difference was nearly negligible in one run (18.0 versus 17.9 requests/s).In contrast, the median increased from 1.6 to 3.2 seconds, as the fast majority of responses disappeared and delays became more evenly distributed. Mean response time decreased from 3.8 to 3.3 seconds. This represents a trade-off: a shorter tail but a longer median.

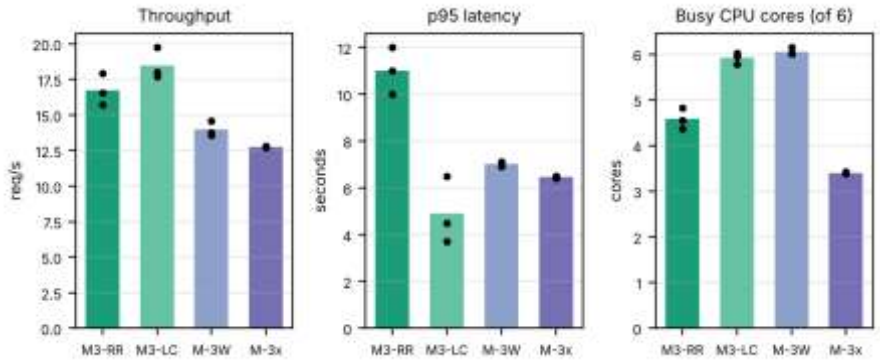


Figure 4. Supplementary experiments at 100 users. Bars show the mean of three runs; points show individual runs. M3-RR: Micro-3 with round-robin scheduling M3-LC : Micro-3-LC; M-3x: Mono-3x ; M-3W: Mono-3W.

Three processes in one container. Mono-3W raised throughput from 12.8 to 14.0 requests/s (approximately 9%) and utilized all six CPU cores (6.1 versus 3.4 for Mono-3x). This result indicates that using a single process had left capacity idle. However, the increase did not approach the throughput of Micro-3 (16.8 requests/s) or Micro-3-LC (18.5 requests/s), because CPU cost per request rose to 0.43 seconds, compared with 0.27 seconds for Mono-3x and 0.32 seconds for Micro-3-LC. In other words, the cores were busy, but this did not translate into a proportional increase in throughput. The cause was not isolated.

Control runs. In this session, Micro-3 and Mono-3x achieved 16.8 and 12.8 requests/s, respectively, compared with 15.3 and 10.7 in the main campaign—approximately 10% and 19% higher, while preserving the same ranking. This supports our caution in interpreting absolute values. The Micro-3 and Micro-3-LC logs recorded a small number of HTTP 502 responses (3 and 5, respectively), with response times ranging from 1 to 74 ms. This is consistent with, but does not prove, the stale-connection-reuse hypothesis.

4.7 Consistency across sessions and tuned single-container variants

Because absolute throughput changed between sessions, we compared Mono-3x and Micro-3 at 100 users in four sessions (Table 5): The ratio of Micro-3 to Mono-3x throughput was 1.43 in the main campaign, 1.31 in the supplementary session, 1.02 in the tuned-baseline session, and 1.16 in a repeat with 200 warm-up requests. In the two later sessions, Micro-3 exceeded Mono-3x in only 5 of 9 paired repetitions, and the 95% intervals overlapped. Mono-3x throughput roughly doubled between the early and the later sessions (from about 11 to about 19–22 requests/s) without any change in configuration; the number of warm-up requests (200 versus 499) did not explain this difference, and we could not identify its cause. The advantage of replication was therefore consistent in direction in the mean at 100 users, but not stable in size, and absolute values should not be compared across sessions. Tail latency showed the opposite pattern: Micro-3 had lower median(0.4-0.8 s versus 2.3-3.3 s)but a higher p95(6.7-7.7 s versus 3.7-4.7 s).

To test whether the single container was handicapped by its single process, two variants with the same budget (6 CPUs, 6 GB) were run in the tuned-baseline session: six single-threaded worker processes (Mono-6W) and three workers with two threads each (Mono-3W2T). Both reached only 12.9-13.7 requests/s at 100 and 150 versus 21.4–22.2 for Mono-3x in the same session, and their CPU cost per request was about three times higher (0.45–0.47 versus 0.16–0.17 s); Mono-6W used 5.1 GiB, because each worker holds its own copy of the model. A possible explanation is contention for memory bandwidth and cache among multiple model copies; in any case, adding processes to the single container did not improve through.

Table 5. Throughput comparison across experimental sessions

#	Session	Warm-up requests	Mono-3x (req/s)	Micro-3 (req/s)	Ratio
1	Main campaign	200	10.7	15.3	1.43
2	Supplementary session	200 (script default)	12.8	16.8	1.31
3	Tuned-baseline session	499	22.2	22.7	1.02
4	Repeat	200	18.8	21.8	1.16

5. Discussion

In the main campaign, the replicated deployment achieved higher throughput than the single six-CPU container under a comparable service budget, while both outperformed the smaller container. However, the improvement varied between sessions (1.02-1.43× at 100 users), absolute throughput fluctuated even without configuration change, and replication had higher p95 latency in later sessions . The results therefore support an average advantage for replication, but they do not justify claiming a fixed or precisely quantifiable speedup. The supplementary experiments explain pat of this: round-robin ignores replica load, and a policy that considers active connections reduced tail latency by more than half while improving throughput, consistent with web-application studies [15] [16]. Replication alone is insufficient; request distribution also matters.

Mono-3x used only about 3.4 of its six available CPU cores, which initially suggested that a single process limit. However, the tuned variants (Mono-3W, Mono-6W, and Mono-3W2T) did not support this: more worker processes lowered throughput per CPU resources. The bottleneck was therefore not simply the single process. A possible cause is contention for memory bandwidth and CPU cache resources among separate copies of the model weights. Which this study did not isolate. Practically, on CPU-only hardware, horizontal replication behind a lightweight proxy matched or exceeded the throughput of a single larger container in the experiments reported here. However, this advantage depended on the request-distribution policy and was not consistent across all performance metrics, and should be interpreted cautiously in line with prior work [19] showing that inference performance depends on deployment architecture and request management as well as on model efficiency.

6. Limitations

- **Hardware and Test Environment: all** measurements were conducted on a single laptop where the load generator and the containers share the same processor making the absolute values specific to this particular device given that the system relies on an intel core i5-13420H processor, with an architecture hybrid performance and efficiency cores and support for simultaneous multithreading the CPU allocations do not necessarily guarantee equivalent computing capacity.
- **Statistical analysis and measurement limitations:** experiments were conducted sequentially for only three replications which compromised independence and produced wide confidence intervals while supplementary experiments were limited to a single workload of 100 users. Additionally , percentiles relied on averages of approximate bucketing in the Locust tool rather than raw data and requests exceeding the 30-second timeout were counted as failures which affected load measurement Accuracy at saturation.
- **Architectural scope and request path:** the comparison focuses on horizontal versus vertical scaling of a single service replicated three times. Micro-3 Behind an Nginx proxy rather than true microservices decomposition. Since the proxy consumes additional resources one CPU core and 256 MB of memory and adds a routing hop the observed differences reflect. The combined effect of replication load balancing and the proxy path together noting that the proxy's impact on latency did not exeed 10 milliseconds and was not tested in front of single containers.
- **Data nature and workload**: the sample comprised 423 short comments averaging 6.8 words on a single topic(Ratbik Lahzi) and was annotated by a single rater without measuring inter-annotator. Agreement this may limit the generalizability of results to longer texts or other domains of the Libyan dialect and the test did not include batch inference or quantization.
- **Open questions:** the study did not isolate the exact reasons for high CPU consumption per request in the MONO-3W configuration or the rare occurrence of HTTP502 errors. Moreover, inference times for the three models were measured sequentially within a single process using untrained classification heads.

7. Conclusion and Future Work

This paper presents an empirical evaluation of an Arabic sentiment analysis service running on a CPU concluding that no measurable differences exist. between deployment configurations under a load of 10 concurrent. users whereas at 50 users and above performance differences became clear. At 100 concurrent users replicated deployment behind a reverse proxy. achieved equal or higher throughput (at a ratio of 1.02 to 1.43). compared to a single six-core container across all four sessions yet the difference varied significantly between sessions and recorded higher latency in later sessions. Conversely the single-core container saturated early and increasing the number of workers within a single container did not improve throughput reaching approximately 13 requests/ s compared to about 22 requests/s for a single worker within the same session. Round Robin produced strong replica imbalance, which least-connections largely removed, raising throughput by around 10% and reduced p95 by more then 50%. On the classification front the accuracy (85.9% and 89.2% across. Cross-validation; 91.8% for the deployed checkpoint on 85 held-out comments) was adequate but was not used to select the deployment

configuration. Because absolute throughput varied up to twofold between sessions, the findings should be read mainly through within*session comparisions.

Future work will investigate the sources of inter-session variability and whether memory-bandwidth contention limits multi-process scaling, consider CPU core pinning on the hybrid CPU, extend the load-balancing evaluation to more workload levels and repetitions, using a separate load-generation machine and network emulation, and add a classical machine-learning, additional hardware, and longer input texts.

Ethical Considerations. The study used publicly available user comments. Personal identifiers were removed during preprocessing, and the data were analyzed only in aggregate form. No attempt was made to identify individual users.

Data and Code Availability

The source code, deployment configurations, load-testing scripts, and experimental artifacts used in this study are publicly available at Zenodo: https://doi.org/10.5281/zenodo.24354639

References

[1] Antoun, W., Baly, F., & Hajj, H. (2020, May). AraBERT: Transformer-based model for Arabic language understanding In Proceedings of the 4th workshop on open-source arabic corpora and processing tools, with a shared task on offensive language detection (pp. 9-15). https://aclanthology.org/2020.osact-1.2/

[2] Abdul-Mageed, M., Elmadany, A., & Nagoudi, E. M. B. (2021). ARBERT & MARBERT: Deep bidirectional transformers for Arabic. Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), 7088-7105. DOI : 10.18653/v1/2021.acl-long.551

[3] Inoue, G., Alhafni, B., Baimukan, N., Bouamor, H., & Habash, N. (2021, April). The interplay of variant, size, and task type in Arabic pre-trained language models. In Proceedings of the sixth Arabic natural language processing workshop (pp. 92-104). https://aclanthology.org/2021.wanlp-1.10/

[4] Al-Haram, S., & Howedi, F. (2025). Analysis of Libyan Dialect Sentiment Using AraBERT and MARBERT: A Comparative Study. Afro-Asian Journal of Scientific Research (AAJSR), 3(4), 175-179. https://aajsr.com/index.php/aajsr/article/view/695

[5] Lamami, R., Arif, M., Saheri, W., Essgaer, M., Agaal, A., & Abuhajar, A. (2026, April). Libyan Restaurants: A New Annotated Corpus for Sentiment Analysis of the Libyan Arabic Dialect DOI : 10.1109/MI-STA68962.2026.11511167

[6] Alfared, R. A., & Aljarm, H. M. (2024). Sentiment analysis of libyan tweets using machine learning algorithms. Malaysian Journal of Industrial Technology, 8(1), 1-12. https://ejournal.unikl.edu.my/index.php/mjit/article/view/643/646

[7] Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., & Stoica, I. (2017). Clipper: A low-latency online prediction serving system. In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17) (pp. 613–627). USENIX Association.

[8] Wang, Y., Wang, Q., & Chu, X. (2022). Energy-efficient online scheduling of transformer inference services on GPU servers. IEEE Transactions on Green Communications and Networking, 6(3), 1649-1659. DOI: 10.1109/TGCN.2022.3171680

[9] Jain, N., Soni, V., Pandey, V. K., & Mishra, N. (2026, March). QuanTool: A Benchmarking Framework for Evaluating Post-Training Quantization with Best Practices for Transformer Models. In 2026 IEEE Madhya Pradesh Section Conference (MPCON) (pp. 563-567). IEEE. DOI: 10.1109/MPCON69668.2026.11508306

[10] Dice, D., & Kogan, A. (2021). Optimizing inference performance of transformers on CPUs. arXiv preprint arXiv:2102.06621. DOI : 10.48550/arXiv.2102.06621

[11] Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. (2015, September). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. 2015 10th Colombian Computing Conference (10CCC) (pp. 583-590). IEEE. 10.1109/ColumbianCC.2015.7333476

[12] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. In Present and ulterior software engineering (pp. 195–216). Springer. https://link.springer.com/chapter/10.1007/978-3-319-67425-4_12

[13] Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., ... Delimitrou, C. (2019). An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 3–18). https://doi.org/10.1145/3297858.3304013

[14] Camilli, M., Janes, A., & Russo, B. (2022). Automated test-based learning and verification of performance models for microservices systems. Journal of Systems and Software, 187, 111225. DOI : 10.1016/j.jss.2022.111225

[15] Zulman, M. R., Husaini, H., Hidayat, R., & Davi, M. (2026). Performance and fault tolerance analysis of load balancing strategies in microservices architecture with controlled failure injection. Sisfo: Jurnal Ilmiah Sistem Informasi, 10(1), 9–18." DOI : 10.29103/sisfo.v10i1.26893

[16] Sulaksono, D. H., Prabiantissa, C. N., Hapsari, R. K., Sirri, L., & RL, D. Y. (2025, September). Implementation of Load Balancing on a Quiz Web Application Using the Least Connection Algorithm with Reverse Proxy Technique. In 2025 7th International Conference on Cybernetics and Intelligent System (ICORIS) (pp. 1-6). IEEE. DOI :10.1109/ICORIS67789.2025.11296059

[17] Pais, S., Cordeiro, J. P., & Jamil, M. (2022). NLP-based platform as a service: A brief review. Journal of Big Data, 9, Article 54. https://doi.org/10.1186/s40537-022-00603-5

[18] Vasireddy, I., Ramya, G., & Kandi, P. (2023). Kubernetes and docker load balancing: State-of-the-art techniques and challenges. International Journal of Innovative Research in Engineering & Management, 10(6), 49-54. DOI: 10.55524/ijirem.2023.10.6.7

[19] Ganiev, A., Chapin, C., De Andrade, A., & Liu, C. C. (2021, June). An architecture for accelerated large-scale inference of transformer-based language models. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers (pp. 163–169).

[20] Little, J. D. (1961). A proof for the queuing formula: L= λ W. Operations research, 9(3), 383-387.

[21] Denning, P. J., & Buzen, J. P. (1978). The operational analysis of queueing network models. ACM Computing Surveys (CSUR), 10(3), 225-261.

[22] Jia, N., Johari, R., Si, N., & Zheng, Z. (2026, August). Experimentation for Different Scheduling Policies on Queues: Mixed Differences-in-Q Estimators Based on Little's Law. In Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2 (pp. 2074-2085).

DOI : 10.1145/3770855.3817634

[23] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long and Short Papers) (pp. 4171–4186).

[24] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems, 32. [papers.nips.cc]

[25] Docker Inc. Docker Documentation. Available at: https://docs.docker.com/ (accessed May/2025)

[26] Tiangolo, S. FastAPI Documentation. Available at: https://fastapi.tiangolo.com/ (accessed May/2025)

[27] NGINX, Inc. NGINX Documentation. Available at: https://nginx.org/en/docs/ (accessed May/2025)

[28] Locust Developers. Locust Documentation. Available at: https://docs.locust.io/ (accessed May/2025)

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of **JIBAS** and/or the editor(s). **JIBAS** and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.