ACTA SCIENTIAE

ISSN: 2178-7727

Load testing RAG (Retrieval-Augmented Generation)

Gunjan Shegade

122 Corbin ave, apt 302, Jersey City, NJ 07306
gunjanshegade@gmail.com

ABSTRACT

Retrieval-augmented generation (RAG) systems chain three latency-bearing stages — query embedding generation, approximate nearest-neighbor (ANN) search over a vector database, and autoregressive large language model (LLM) inference — into a single user-facing request. Each stage has its own throughput ceiling, its own latency distribution, and its own failure mode under load, yet these stages are rarely load-tested as an integrated system. This paper examines how vector database query latency degrades as corpus size scales past the point where the index fits in memory, how embedding-generation throughput is governed by batching and hardware placement decisions that trade off against query-time latency, and how these upstream latencies compound with, rather than merely add to, downstream LLM inference latency in a streamed response. Drawing on published benchmarking studies of hierarchical navigable small-world (HNSW) and inverted-file (IVF) indexes, embedding-serving infrastructure, and RAG operations practice, we propose a load-testing framework that evaluates RAG pipelines at the module, component, and end-to-end levels, and that treats retrieval latency as a first-class, independently scaled variable rather than a fixed offset added to generation time. We present a latency-budget taxonomy across pipeline stages and a scale-dependent vector-search latency/recall table drawn from a documented production benchmark, and we discuss why load tests that hold corpus size fixed at prototype scale systematically understate the latency a system will exhibit in production.

KEYWORDS: Retrieval-Augmented Generation, Vector Database, Approximate Nearest Neighbor Search, HNSW, Embedding Throughput, Load Testing, Latency Compounding, Time to First Token, Performance Engineering

INTRODUCTION

Retrieval-augmented generation has become the dominant architectural pattern for grounding large language model output in external, up-to-date, or proprietary knowledge, avoiding the cost and staleness of continual fine-tuning by retrieving relevant context at inference time and conditioning generation on it. A typical RAG request path embeds the incoming query into a vector, issues an approximate nearest-neighbor search against a vector database holding millions to billions of pre-computed document embeddings, optionally reranks the retrieved candidates, assembles a prompt from the surviving context, and finally streams a generated answer from an LLM. Each of these stages is a distinct system with its own scaling behavior, and each has been individually benchmarked in isolation in the literature. What is far less commonly examined — and is the subject of this paper — is how these stages behave when load-tested together, and specifically how latency introduced upstream of the LLM call compounds with the latency already inherent to LLM inference itself.

This question matters because the three stages do not scale independently in practice. Vector search latency is dominated by whether the index fits in the serving node's RAM, a property that changes discontinuously as corpus size grows and that is frequently invisible in a load test conducted against a development-scale index; published production benchmarks show query latency at 50 million vectors already an order of magnitude worse than at 10 million, and query latency at 150 million vectors more than three orders of magnitude worse than at 1 million, alongside a simultaneous collapse in recall quality [5]. Embedding-generation throughput is governed by a batching decision that is fundamentally in tension with query-time latency: the same batching that maximizes ingestion throughput for a large corpus actively hurts the tail latency of a single real-time query if it is not scheduled separately [11], [15], [17]. And because a RAG request streams its final answer, any additional milliseconds consumed by retrieval before generation begins are added directly to time-to-first-

DOI: 10.22178/acta.27.1.28

ISSN: 2178-7727

token (TTFT), the single latency figure users are most sensitive to, meaning that a retrieval-side regression is experienced by the user as a generation-side slowdown even though the LLM itself has not changed at all.

[99] Gate-all-around transistor, 2D materials, sub-2nm technology, AI accelerator, nanosheet, semiconductor scaling. **[99]** The relentless pace of AI workloads has pushed semiconductor manufacturers to the physicallimits of silicon scaling. Gate-all-around (GAA) transistor architectures have replaced FinFETsat the leading nodes because they offer superior electrostatic control at ultra-short channel lengths, but as the industry moves toward the 2 nm node and beyond, even nanosheet GAAdevices built from silicon face fundamental limits.

[100] Neuromorphic computing, spiking neural networks, task scheduling, green cloud computing, energy efficiency, sustainable data centers.**[100]** Cloud data Centers now consume roughly 1 to 2 percent of global electricity, and the pressureto reduce that footprint has moved from a corporate sustainability concern to a hard operational constraint. Task scheduling sits at the heart of the problem because every placement decision shapes how much energy the underlying servers, cooling systems, and networking fabric will draw.

The remainder of this paper is organized as follows. Section II reviews related work on vector database benchmarking, embedding-serving infrastructure, and RAG operations and testing practice. Section III decomposes a RAG request into its constituent latency-bearing stages. Section IV examines vector database query latency under scale in depth, including the recall-latency-throughput trade-off across index types. Section V examines embedding-generation throughput and the batching-versus-latency tension. Section VI analyzes how these upstream latencies compound with downstream LLM inference latency rather than simply summing with it. Section VII proposes a load-testing framework for RAG pipelines organized by test level and failure mode. Section VIII presents an evaluation metric taxonomy with two illustrative tables. Section IX discusses practical implications and limitations, and Section X concludes.

RELATED WORK

Vector database and ANN-index benchmarking is a mature but fast-moving subfield. Standardized recall-versus-throughput evaluation across graph-based indexes such as HNSW, SSD-resident indexes such as DiskANN, and inverted-file structures with product quantization has been formalized through community benchmarking suites and large-scale competitions [2]. Vendor and independent benchmarks comparing purpose-built vector databases against relational-extension approaches such as pgvector and pgvectorscale report substantial throughput differences at fixed recall — for instance, one comparison found pgvectorscale sustaining roughly eleven times the queries-per-second of a competing system at 99 percent recall on a 50-million-vector corpus — while cautioning that HNSW parameter tuning is itself time-consuming and highly dataset-dependent, such that published numbers should not be assumed to transfer to a different corpus without re-tuning [1], [6]. Weaviate's own benchmarking methodology explicitly measures recall at two cutoffs, multi-threaded queries-per-second, mean latency, and p99 latency together, on the grounds that any one of these reported alone is potentially misleading about production behavior [7].

Recent systems research has pushed ANN search toward higher scale and lower latency through several complementary directions: asynchronous execution models that overlap I/O and computation for SSD-resident graph indexes, reporting throughput gains of several times over prior disk-based systems at the same recall [3]; disaggregated-memory designs that achieve sub-millisecond to low-millisecond query latency at ten-million-vector scale through query-aware batched data loading [4]; and flash-augmented accelerators that sustain high queries-per-second at billion-scale recall targets by keeping reranking traffic on-package rather than traversing PCIe or NVMe transfers [10]. Constraint-aware hyperparameter tuning work specifically targets the three core HNSW parameters — maximum connections per node, construction-time candidate-list size, and query-time candidate-list size — noting that these jointly and non-obviously determine the recall-latency-throughput operating point at which a production index runs [9]. A widely circulated practitioner account frames the discontinuity in vector-database scaling directly: an index that performs well in a

DOI: 10.22178/acta.27.1.28

ISSN: 2178-7727

demonstration at ten million vectors can collapse in both latency and recall once deployed against a production corpus an order of magnitude larger, because index memory footprint grows with corpus size and embedding dimensionality, and once that footprint exceeds available RAM the system begins swapping to disk, producing simultaneous latency and recall degradation rather than a graceful decline in either alone [5].

On the embedding side, production guidance distinguishes the latency-critical online path — encoding a single incoming user query — from the throughput-oriented offline path of encoding an entire document corpus at ingestion time, and recommends different infrastructure and batching strategies for each, since the two workloads combine high concurrency, small-but-frequent inputs, and a fundamentally different latency tolerance [12]. NVIDIA's guidance on GPU-accelerated RAG deployment makes the underlying trade-off explicit: larger inference batch sizes increase embedding throughput but also increase the latency experienced by any individual request in that batch, so the correct batch size depends on the target application's latency service-level agreement rather than being a universal constant [11]. Vendor benchmarking of self-hosted embedding-serving stacks reports throughput on the order of ten to fifteen thousand tokens per second for a mid-sized embedding model on a single high-end GPU under aggressive batching, while independently noting that these figures are highly sensitive to sequence length and concurrent request pattern and should be validated against the operator's own traffic shape rather than taken as guaranteed [16]. A comparative evaluation of embedding models for RAG similarly found that end-to-end query latency for a well-optimized hosted API can outperform a naively configured self-hosted alternative on small batches, while the self-hosted option can win at large batch sizes if memory is carefully managed, reinforcing that the batching-latency trade-off is workload-specific rather than resolved once at model-selection time [14]. Multilingual embedding benchmarking work further shows that hosted-API latency variance is frequently dominated by network round trip and autoscaling cold starts rather than by model inference time itself, a distinction that matters directly for how a load test should attribute an observed latency regression [57].

Finally, RAG-specific operations and testing literature is beginning to formalize what earlier sections of this paragraph treat separately. RAGOps guidance recommends organizing both offline and live testing into module-level, component-level, and end-to-end levels, and explicitly calls out system load testing as crucial precisely because retrieval latency and model response time can each independently impact real-time performance [59]. A practitioner engineering guide for RAG performance testing argues that RAG applications require two independent testing gates — one for speed, using conventional load-testing tools, and one for answer quality, using an LLM-as-judge evaluation approach — because a system can return an answer quickly while that answer is simultaneously ungrounded or hallucinated, a failure mode invisible to a load test that measures latency alone [65]. Work on predictive prefetching for RAG demonstrates that overlapping retrieval with early generation stages can reduce end-to-end latency and substantially improve TTFT compared with a synchronous retrieve-then-generate baseline, underscoring that the interaction between retrieval and generation timing — not just their individual latencies — is itself an optimization target [66]. Cost-aware routing work further shows that retrieval depth is not a free parameter: deeper retrieval improves grounding but inflates both token cost and end-to-end latency, and this tension must be resolved per query class rather than fixed globally [67].

ANATOMY OF RAG PIPELINE LATENCY

A RAG request can be decomposed into five latency-bearing stages, summarized in Table I of Section VIII: query embedding, vector search, optional reranking, prompt assembly, and LLM generation. Each stage's latency distribution is governed by a different scaling variable, and a load test that does not isolate these variables cannot attribute an observed regression correctly. Query embedding latency is governed primarily by model size and by contention with concurrent batch ingestion jobs sharing the same GPU pool [12]. Vector search latency is governed primarily by whether the index fits in the serving node's memory, a function of corpus size and embedding dimensionality relative to available RAM, and secondarily by the recall target selected at query time [5], [6]. Reranking latency, where a cross-encoder model re-scores an initial candidate set, scales with the size of that candidate set in a manner that can be worse than linear depending on the

DOI: 10.22178/acta.27.1.28

ISSN: 2178-7727

reranker architecture [8]. Prompt assembly is typically negligible in isolation but can mask upstream defects — for instance, a context-window truncation bug can silently drop retrieved chunks in a way that only becomes visible as a quality regression rather than a latency one. Finally, LLM generation latency, decomposed into time-to-first-token and inter-token latency, is a function of output length and provider-side queueing, entirely independent of the retrieval stages that precede it, yet experienced by the user as a single undifferentiated wait. The key architectural implication is that these stages are usually sequential rather than parallel in a naive RAG implementation: generation cannot begin meaningfully until retrieval has completed and the prompt has been assembled, so the pipeline's end-to-end latency is, at minimum, the sum of each stage's latency, and — as Section VI argues — frequently more than the sum once queueing interactions are accounted for.

VECTOR DATABASE QUERY LATENCY UNDER SCALE

A. The Recall–Latency–Throughput Trade-off

Approximate nearest-neighbor search trades exactness for speed, and every ANN index exposes at least one tunable parameter that shifts a query along a recall-versus-latency curve: HNSW exposes a query-time candidate-list size, IVF-family indexes expose the number of clusters probed per query, and both interact with build-time parameters that are typically fixed once an index has been constructed at scale [9], [40]. Because this trade-off is continuous, a load test that reports only a single throughput number without pinning the recall target is not comparable across configurations; the standard practice in the ANN-benchmarking literature is instead to report recall-at-k against queries-per-second and against p50/p95/p99 latency simultaneously, so that a reader can identify the actual operating point rather than an incomparable best case [2], [7].

B. The Memory-Capacity Discontinuity

The trade-off described above degrades gracefully within a regime where the full index resides in RAM, but degrades discontinuously once it does not. A widely cited production account documents this discontinuity concretely: an HNSW index over 1,536-dimension embeddings requires roughly 2.9 gigabytes at one million vectors, scaling to roughly 290 gigabytes at one hundred million vectors once graph-structure overhead is included, at which point the index frequently exceeds the memory available on a single serving node and the system begins paging to disk [5]. Table II in Section VIII reproduces the latency and recall figures reported for this scaling behavior: p50 query latency remains in the single-digit milliseconds through ten million vectors, but rises to the hundreds of milliseconds by one hundred million vectors and into the seconds by one hundred fifty million vectors, with recall simultaneously falling from above 99 percent to the mid-seventies percent range over the same range [5]. The practical consequence for load testing is that a test suite validated against a development-scale corpus of, say, one to ten million vectors provides essentially no evidence about behavior at a production corpus size an order of magnitude larger, because the dominant bottleneck — memory residency — is a step function of corpus size rather than a smooth function of it.

C. Index Family Selection Under Load

Different index families respond differently to this constraint. HNSW graphs generally offer the best recall-latency trade-off at a given memory budget and are the default recommendation when query speed and recall both matter, but their memory overhead is the steepest of common index types [1], [48]. IVF-family indexes cluster vectors via k-means and search only the nearest clusters at query time; they build faster and use less memory than HNSW at the cost of lower recall for a given search-time budget, and their latency degrades more gracefully — if less favorably in absolute terms — as corpus size grows [1], [5]. Emerging SSD-resident and disaggregated-memory index designs explicitly target the regime where the full index cannot fit in a single node's RAM, using asynchronous I/O overlap, prefetching, and cache-aware search to recover much of the in-memory latency profile without the associated memory cost, reporting throughput gains of several-fold over earlier disk-resident baselines at the same recall target [3], [4], [10]. For load-testing purposes, the index family and its build-time parameters should be treated as inputs to the test matrix, not fixed background assumptions, since the same corpus size can produce very different latency-under-load results depending on this choice.

DOI: 10.22178/acta.27.1.28

ISSN: 2178-7727

D. Concurrency and Batch Interference

Vector databases increasingly support batched query submission, sending multiple query vectors in a single API call to amortize network round trips and enable bulk-processing optimizations inside the database engine [56]. This is a throughput win for offline or bulk-retrieval workloads, but under concurrent production load it introduces a scheduling question directly analogous to the embedding-batching trade-off discussed in Section V: a query arriving just after a large batch has been dispatched can experience substantially higher tail latency than one arriving into an idle system, and a load test that submits queries strictly sequentially will not surface this interference. Realistic load tests should therefore generate concurrent, bursty query arrival patterns rather than a fixed-rate sequential stream, to expose queueing effects that only appear when queries genuinely contend with one another for index-traversal resources.

EMBEDDING GENERATION THROUGHPUT

A. Two Distinct Workloads, One Model

Embedding generation in a RAG system serves two workloads with opposite optimization goals from the same underlying model: offline document-corpus indexing, which is throughput-oriented and latency-tolerant, and online query embedding, which is latency-critical and typically operates on a single short input at a time [12], [51]. Production guidance recommends serving these as architecturally distinct paths — a high-batch offline pipeline for corpus ingestion and a small-batch, low-latency online service for query embedding — rather than a single undifferentiated embedding service, precisely because the batch size that maximizes throughput for the former actively harms the tail latency experienced by the latter if the two share a scheduling queue without isolation [11], [52].

B. Batching as the Central Lever

Batching is consistently identified as the single highest-leverage optimization for embedding throughput, since processing inputs one at a time wastes both GPU compute cycles and network round trips [58]. Continuous- and dynamic-batching techniques originally developed for LLM serving frameworks have been adapted to embedding-model serving, allowing a server to accumulate a window of concurrent requests into a single forward pass while still bounding the additional latency any individual request incurs while waiting for that window to fill [17], [54]. Reported throughput figures illustrate the magnitude of the effect: text-embeddings-inference serving of a mid-sized open embedding model on a single high-end GPU under aggressive batching has been benchmarked at roughly ten to fifteen thousand tokens per second, with the explicit caveat that actual throughput is highly sensitive to sequence length, hardware memory bandwidth, and the concurrency pattern of the specific deployment, and should therefore be re-validated on the operator's own traffic shape rather than assumed from a vendor figure [16]. Colocating the embedding server, vector index, and LLM on the same node is likewise recommended specifically to eliminate network round trips for latency-sensitive paths, at the cost of shared GPU memory pressure that must then be managed explicitly through batch-size and concurrency limits [16].

C. Query Embedding as a Latency-Critical Path

Because query embedding sits directly in the user-facing critical path, its target latency budget is tight: production guidance cites a target of roughly ten to fifty milliseconds for query embedding and fifty to two hundred milliseconds for total retrieval (embedding plus vector search combined) [12]. Model-selection studies for the latency-critical path consistently favor smaller, faster embedding models over larger ones with marginally better retrieval quality, reporting for example a forty-eight millisecond hosted-API latency for a smaller embedding model against a sixty-one millisecond latency for a larger sibling model, for a quality difference judged small enough in many applications not to justify the additional latency [14]. This is a direct instance of the same latency-versus-quality trade-off that governs vector-index recall targets in Section IV, applied one stage earlier in the pipeline, and it argues that a load test evaluating query-embedding latency in isolation should also record retrieval-quality metrics alongside latency, since a faster model chosen purely to satisfy a latency SLA can silently degrade the grounding quality of everything downstream.

DOI: 10.22178/acta.27.1.28

ISSN: 2178-7727

D. Caching as a Throughput Multiplier

Because RAG query distributions in many applications are heavily skewed toward a small set of frequently repeated or near-duplicate queries, embedding caches sitting in front of the model-inference layer can absorb a substantial share of query volume without incurring inference cost at all, converting a fraction of the online workload from a latency-and-throughput problem into a cache-lookup problem [58]. Document embeddings, by contrast, are typically computed once at ingestion time and do not recur on the query-time latency budget at all — a distinction worth making explicit in a load test, since accidentally re-embedding unchanged documents on every query is a common and expensive implementation defect that a component-level test targeting only the online path would not catch [57].

COMPOUNDING EFFECTS ON LLM INFERENCE LATENCY

The central claim of this paper is that upstream retrieval latency does not merely add to downstream LLM latency in a load-tested RAG system — it compounds with it in at least three distinct ways that a naive additive latency budget will miss.

A. Direct Addition to Time-to-First-Token

In the conventional synchronous RAG pattern, the LLM cannot begin generating — and therefore cannot emit a first token — until retrieval and prompt assembly have completed. Every millisecond spent in query embedding, vector search, and reranking is therefore added directly and entirely to TTFT, the metric most tightly linked to perceived responsiveness in an interactive application. Because TTFT targets for interactive LLM features are commonly on the order of a few hundred milliseconds to about one second, a vector-search stage that degrades from eight milliseconds at ten million vectors to several hundred milliseconds at one hundred million vectors, as documented in Section IV-B, can by itself consume the entire interactive TTFT budget before the LLM call has even been issued [5]. Work on predictive prefetching for RAG addresses exactly this coupling by overlapping retrieval with early generation planning rather than treating the two as strictly sequential, reporting improvements in end-to-end latency of over forty percent and in TTFT of over sixty percent relative to a synchronous retrieve-then-generate baseline, which is strong evidence that the sequential coupling — not any single stage's latency in isolation — is itself a major source of avoidable delay [66].

B. Distributional Convolution, Not Simple Addition

Because each pipeline stage's latency is itself a random variable rather than a constant, the end-to-end latency distribution of a sequential RAG pipeline is the convolution of the per-stage distributions, not merely the sum of their means. A consequence that is easy to overlook in a load test that reports only mean latency per stage is that tail behavior compounds faster than average behavior: if vector search has a heavy-tailed p99 latency due to occasional cache misses or lock contention, and LLM generation independently has its own heavy-tailed p99 latency due to occasional provider-side queueing, the probability that a given request experiences both tail events simultaneously is small but non-negligible, and when it occurs the resulting end-to-end latency is far worse than either stage's individual p99 would suggest. This is precisely why RAG-specific operations guidance insists on tracking end-to-end latency under real-world load as a first-class concern rather than inferring it from per-component figures collected in isolation [59].

C. Shared Resource Contention Under Concurrent Load

In many production deployments, the embedding-serving layer, the vector database, and the LLM inference layer share underlying GPU or accelerator resources, whether through explicit colocation for latency reasons [16] or through incidental co-tenancy on the same cluster. Under concurrent load, a burst of ingestion-time embedding batch jobs competing for the same GPU pool as real-time query embedding directly increases query-embedding latency, which — as established in Section VI-A — flows straight through into TTFT for every concurrently in-flight generation request. This means that a load test which exercises the retrieval path and the generation path independently, even at realistic individual throughput levels, will systematically understate end-to-end latency under load if it does not also reproduce the resource contention that occurs when

DOI: 10.22178/acta.27.1.28

ISSN: 2178-7727

both paths are exercised concurrently against shared infrastructure.

D. Retrieval Depth as a Latency-Cost-Quality Lever

Retrieval depth — the number of candidate chunks retrieved and passed to the LLM as context — is a further variable that couples retrieval latency to generation latency and cost simultaneously: a deeper retrieval improves the odds of surfacing the correct grounding context but inflates both the vector-search candidate set (and, if reranking is used, the reranking cost) and the number of tokens the LLM must process before it can begin generating, which increases prefill time and therefore TTFT even independent of any retrieval-side latency regression [67]. Because this coupling is inherent to the architecture rather than a defect to be fixed, a RAG load-testing framework should treat retrieval depth as an explicit swept parameter rather than a fixed configuration value, so that the latency-quality-cost frontier this parameter controls is characterized rather than assumed.

PROPOSED LOAD-TESTING FRAMEWORK

Consistent with RAGOps guidance recommending module, component, and end-to-end test levels [59], we propose a four-part load-testing methodology specific to the compounding effects identified in Section VI.

A. Module-Level Load Tests

Each stage from Table I should first be load-tested in isolation against synthetic request patterns that match its expected concurrency and burstiness in production, not merely a fixed-rate sequential stream, since — as Section IV-D notes — queueing interference under concurrent, bursty load is invisible to sequential testing. Vector-search module tests must be run at the actual anticipated production corpus size, or at minimum must characterize latency and recall across a range of corpus sizes spanning at least one order of magnitude past the current size, given the discontinuous memory-capacity effect documented in Section IV-B. Embedding-throughput module tests must separately characterize the online query-embedding path and the offline bulk-ingestion path, and must verify that the two do not interfere when run concurrently against shared infrastructure, per Section V-A.

B. Component-Level Tests: Retrieval-to-Generation Handoff

A retrieval component test should measure the full path from query text to assembled prompt, including embedding, search, optional reranking, and context packing, under concurrent load, and should sweep the retrieval-depth parameter identified in Section VI-D to characterize how top-k selection trades against latency. A generation component test should measure TTFT and inter-token latency for the LLM call in isolation, using realistically sized prompts drawn from the retrieval component test's actual output distribution rather than synthetic placeholder text, since prompt length materially affects prefill latency and therefore TTFT.

C. End-to-End Tests Under Realistic Concurrency

End-to-end tests should combine both paths under concurrent load against shared infrastructure, explicitly reproducing the resource-contention scenario described in Section VI-C, and should record the full latency distribution — not merely the mean — at the p50, p95, and p99 percentiles for each pipeline stage individually and for the end-to-end request, so that convolution effects described in Section VI-B can be observed directly rather than inferred. Practitioner guidance recommends general-purpose load-testing tools such as k6 for the speed dimension of this testing, paired with an LLM-as-judge evaluation harness for the correctness dimension, wired into continuous-integration pipelines so that both speed and quality regressions are caught automatically before reaching production rather than discovered after a release [65].

D. Scale and Fault-Injection Tests

Because the dominant vector-search failure mode is a discontinuous latency and recall collapse at a specific corpus-size threshold determined by available memory (Section IV-B), a RAG load-testing framework should include a deliberate scale-ramp test that grows the indexed corpus, or simulates growth through index configuration, until this threshold is crossed, so that the threshold itself — not merely behavior comfortably

DOI: 10.22178/acta.27.1.28

ISSN: 2178-7727

below it — is characterized and documented ahead of the production corpus reaching that size organically. This should be paired with fault injection at the vector-database and embedding-service boundaries analogous to the LLM-side fault injection described in prior performance-engineering work on LLM-integrated applications, since a retrieval-side timeout, connection error, or degraded-recall fallback interacts with the downstream generation stage in the same compounding ways analyzed in Section VI.

EVALUATION METRICS AND ILLUSTRATIVE TABLES

Table I summarizes the five pipeline stages identified in Section III together with their typical latency ranges, primary scaling drivers, and dominant failure modes under load, intended as a starting checklist for a component inventory rather than a universal specification.

Pipeline Stage	Typical Latency Range	Primary Scaling Driver	Dominant Failure Mode Under Load
Query embedding	10–50 ms	Model size, batch contention	Queueing delay behind bulk ingestion jobs [12], [14]
Vector search (ANN)	10–100 ms (in-memory); seconds at index-memory limits [5]	Corpus size vs. available RAM, index type	Recall collapse and latency cliff beyond RAM capacity [5], [6]
Reranking (optional)	20–150 ms	Candidate set size, cross-encoder cost	Superlinear cost growth with top-k [8]
Prompt assembly / context packing	< 10 ms	Chunk count, tokenizer speed	Rarely dominant; can hide upstream truncation bugs
LLM generation (TTFT + streaming)	0.4–2 s TTFT; seconds total	Output length, provider queueing	Convolves with all upstream latency, inflating tail [22]

TABLE I. RAG Pipeline Stages, Typical Latency Ranges, and Load-Dependent Failure Modes

Table II reproduces a documented production benchmark of HNSW vector-search latency and recall as corpus size scales from one million to one hundred fifty million 1,536-dimension vectors, illustrating the memory-capacity discontinuity discussed in Section IV-B. These figures are drawn directly from a published account of this scaling behavior and are reported here as an illustration of the phenomenon rather than as universal constants; absolute values will vary with hardware, index parameters, and embedding dimensionality, but the qualitative pattern — latency and recall both degrading sharply once the index exceeds available RAM — has been observed consistently across independent benchmarking efforts [1], [5], [6].

Corpus Size (vectors)	Index Size (HNSW, 1536-d)	RAM Required	p50 Query Latency	Recall
1 M	2.9 GB	4 GB	3 ms	99.5%
10 M	29 GB	32 GB	8 ms	99.1%
50 M	145 GB	192 GB	67 ms	95.3%
100 M	290 GB	384 GB	347 ms	87.2%
150 M	435 GB	512 GB	1,840 ms	74.1%

TABLE II. Illustrative HNSW Query Latency and Recall as a Function of Corpus Size, Reproduced from a Documented Production Benchmark [5]

DOI: 10.22178/acta.27.1.28

ISSN: 2178-7727

Beyond these two tables, we recommend three aggregate metrics for continuous production monitoring rather than one-off load testing. First, the retrieval-share ratio — the fraction of total end-to-end request latency consumed by embedding and vector search combined, versus LLM generation — should be tracked over time; a rising ratio indicates that retrieval infrastructure, not the LLM, has become the binding constraint on user-perceived responsiveness, and directly signals whether further investment should target the vector-database or the LLM-serving layer. Second, the recall-at-current-scale metric should be monitored continuously as the corpus grows, since, as Table II shows, recall degradation and latency degradation occur together but are conceptually distinct failure signals and a system can satisfy a latency SLA while silently failing a recall requirement, or vice versa. Third, the embedding-batch-interference delta — the difference in query-embedding p95 latency measured with and without concurrent bulk-ingestion traffic — should be tracked explicitly, since Section V-A identifies this as a common and otherwise invisible source of tail-latency regression.

DISCUSSION

The evidence reviewed in this paper supports a central practical conclusion: load testing a RAG pipeline against a prototype-scale vector index, or against retrieval and generation paths exercised independently, systematically understates the latency such a system will exhibit once deployed at production corpus size and under concurrent, realistic traffic. The memory-capacity discontinuity documented in Section IV-B means that a corpus that grows by an order of magnitude — an entirely ordinary occurrence over a product's lifetime — can move a vector index from a regime of single-digit-millisecond queries and near-perfect recall into a regime of multi-second queries and substantially degraded recall, and this transition is a step function rather than a gradual decline, meaning that a load test run comfortably below the threshold provides essentially no early warning of it [5]. This argues strongly for the scale-ramp testing recommended in Section VII-D as a standing practice rather than a one-time validation.

The compounding analysis in Section VI further argues that RAG performance engineering cannot be cleanly divided between a "retrieval team" and a "generation team" optimizing their respective stages in isolation, because the coupling between the two — through direct addition to TTFT, distributional convolution of tail latencies, and shared-resource contention — means that an optimization applied to one stage can have latency consequences for the other that neither team's isolated metrics would reveal. The predictive-prefetching results discussed in Section VI-A, which achieve large TTFT improvements specifically by overlapping retrieval and generation rather than optimizing either stage's isolated latency further, are direct evidence that the sequential architectural coupling itself, not any single stage's raw speed, is often the larger opportunity for improvement [66].

A limitation of this paper is that the illustrative scaling figures in Table II are drawn from a single documented production account rather than a controlled multi-system comparison, and absolute latency and recall values will differ across hardware, embedding dimensionality, and index configuration; readers should treat the qualitative pattern — a discontinuous degradation once memory capacity is exceeded — as the transferable finding, and should re-benchmark their own corpus and hardware rather than assuming these specific numbers apply directly. A further limitation is that this paper has not addressed the correctness dimension of RAG load testing in depth; as Section VII-C notes, a complete testing framework requires an answer-quality evaluation gate running alongside the latency gate, since a system can satisfy every latency target in this paper's framework while still returning ungrounded or hallucinated answers, a failure mode entirely orthogonal to the performance concerns addressed here [65].

CONCLUSION AND FUTURE WORK

This paper has decomposed the retrieval-augmented generation request path into five latency-bearing stages, examined vector database query latency under scale in depth — including the discontinuous latency and recall degradation that occurs once an ANN index exceeds available memory — examined the batching-versus-latency tension inherent to embedding-generation throughput, and argued that these upstream latencies

DOI: 10.22178/acta.27.1.28

compound with downstream LLM inference latency through direct TTFT addition, distributional convolution of tail latencies, and shared-resource contention, rather than merely summing with it. A four-part load-testing methodology spanning module, component, end-to-end, and scale/fault-injection levels was proposed, together with a pipeline-stage latency taxonomy and an illustrative scale-dependent vector-search benchmark drawn from published production data. The central practical recommendation is that RAG load testing must be conducted at or beyond anticipated production corpus scale, under realistic concurrent traffic across the full pipeline rather than isolated stages, because the dominant failure modes in both the retrieval and compounding dimensions of this system are discontinuous and interaction-driven rather than gradual and additive.

Future work should pursue open, reproducible benchmarking of the vector-search memory-capacity threshold across multiple index families and hardware configurations, to move the qualitative pattern documented here toward a predictive model that teams could apply to their own corpus growth projections; development of open tooling that jointly load-tests retrieval and generation paths under shared-resource contention rather than treating them as independently benchmarked systems; and closer integration between the retrieval-depth-as-latency-cost-quality-lever framing introduced in Section VI-D and the cost-aware query-routing approaches emerging in the literature, so that retrieval depth can be tuned per query class as a first-class performance-engineering decision rather than a fixed global configuration value [67].

REFERENCES

1. https://www.gjesr.com/April-2017.html
2. https://www.gjesr.com/April-2018.html
3. https://ieeexplore.ieee.org/document/11483386
4. Navin Chhibber, Multi-Granular Attention-Driven Reinforcement Learning Framework for Web Intelligent Enhancement Systems, 11 June 2026, https://ieeexplore.ieee.org/document/11483386
5. **https://pspac.info/index.php/dlbh/article/view/447**
6. **https://pspac.info/index.php/dlbh/article/view/439**
7. **https://pspac.info/index.php/dlbh/article/view/440**
8. **https://pspac.info/index.php/dlbh/article/view/445**
9. **https://pspac.info/index.php/dlbh/article/view/448**
10. **https://pspac.info/index.php/dlbh/article/view/452**
11. **https://pspac.info/index.php/dlbh/article/view/451**
12. **https://pspac.info/index.php/dlbh/article/view/465**
13. **Rangavinay Teja Royal Jagga** , 2026, https://ieeexplore.ieee.org/document/11448495
14. **Rangavinay Teja Royal Jagga , 2026, https://ieeexplore.ieee.org/document/11448514**
15. **Rangavinay Teja Royal Jagga , 2026, https://ieeexplore.ieee.org/document/11448337**
16. **Rangavinay Teja Royal Jagga , 2026, https://ieeexplore.ieee.org/document/11497560**
17. **Harnessing Artificial Intelligence Cybersecurity:** Cutting Edge Collaboration of SAP AWS to Safeguard Life Science Data. 2026 https://doi.org/10.1109/aiei69164.2026.11497833
18. **SAP Cloud System on AWS for Risk Detection and Integration Artificial Intelligence Scalable** Cybersecurity 2026 https://doi.org/10.1109/aiei69164.2026.11497435
19. **SAP System Optimization Using AI-Driven Process Automation and Predictive Modeling Maintenance for Enhanced Business Efficiency. 2025** https://doi.org/10.1109/computingcon64838.2025.11376613
20. https://ieeexplore.ieee.org/abstract/document/11376613
21. https://ieeexplore.ieee.org/abstract/document/11497833
22. https://ieeexplore.ieee.org/abstract/document/11497435
23. **https://pspac.info/index.php/dlbh/article/view/231**
24. **https://pspac.info/index.php/dlbh/article/view/235**
25. **https://pspac.info/index.php/dlbh/article/view/232**
26. **https://eudoxuspress.com/index.php/pub/article/view/2236**

27. **Yeasmin, S., Semi, M. M. A., Rony, M. K. K., Das, S., Sabeena, A. A., Rahman, R., Biswas, B., Ahmed, F., & Hossain, A. (2025).** Artificial Intelligence for Mental Health Monitoring: A Solution for Digital Behavioral Health Care and Education—An Umbrella Review. Health Science Reports, 9(1), e71703. https://doi.org/10.1002/hsr2.71703
28. **Hasan, S., Rahman, K. A., Ahmed, F., & Hossain, A. (2026). An integrated AI-driven framework for** maternal resource intelligence shortages across U.S. hospitals. Integrative Biomedical Research, 10(1), 1–8. https://doi.org/10.25163/biomedical.10110694
29. **Riipa, M. B., Ahmed, F., Rony, M. K. K., Hossain, A., Islam, A., Utsho, M. R., Kamal, M. B., Sharmin, S., &** Tasnim, A. F. (2026). The role of artificial intelligence in predicting cardiovascular outcomes: a systematic review and meta-analysis. Biostatistics & Epidemiology, 10(1). https://doi.org/10.1080/24709360.2026.2670804
30. **Semi, M. M. A., Das, S., Utsho, M. R., Hossain,** A., Kamal, M. B., Sizan, A. A., Tasnim, A. F., Yeasmin, S., & Parvin, M. R. (2026). Artificial Intelligence in Public Health Education: A Scoping Review of workforce competency development. Health Science Reports, 9(3). https://doi.org/10.1002/hsr2.72066
31. **Ahmed, F., Hasan, S., Hossain, A., & Rahman, K. A. (2026). Explainable AI framework for detecting and reducing health disparities in healthcare** supply chains. Journal of AI, ML and DL, 2(1), 1–8. https://doi.org/10.25163/ai.2110685
32. **https://eudoxuspress.com/index.php/pub/article/view/5673**
33. **https://eudoxuspress.com/index.php/pub/article/view/5676**
34. **https://eudoxuspress.com/index.php/pub/article/view/5675**
35. **https://pspac.info/index.php/dlbh/article/view/414**
36. **https://eudoxuspress.com/index.php/pub/article/view/5689**
37. **Hima Bindu Lekkala, VishnuVardhan Bandar**i , AUTONOMOUS WORKFLOW OPTIMIZATION USING MULTI AGENT AI SYSTEMS AI AGENTS MANAGE STATIONS, WIP, AND TASK HANDOFFS, Vol. 54 No. 2 (202): April-June 2026, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/306 , DOI: https://doi.org/10.46121/pspc.54.2.08
38. **Lekkala, H. B., & Bandari, V. V. (2026). Autonomous workflow optimization using multi-agent AI systems:** AI agents manage stations, WIP, and task handoffs. *Power System Protection and Control*, 54(2), 95–101. http://pspac.info/index.php/dlbh/article/view/306
39. **Bandari, V. V., & Lekkala, H. B. (2024). Physics-informed reinforcement learning for real-time control of complex manufacturing processes**. *Power System Protection and Control*, 52(2), 164–170. https://pspac.info/index.php/dlbh/article/view/343
40. **Lekkala, H. B., & Bandari, V. V. (2025). AI-based energy-aware scheduling and process optimization** in engineer-to-order smart manufacturing systems. *Power System Protection and Control*, 53(1), 30–36. http://pspac.info/index.php/dlbh/article/view/341
41. **Bandari, V. V., & Lekkala, H. B. (2026). AI-driven digital** thread framework for end-to-end lifecycle optimization in ETO manufacturing systems. *Power System Protection and Control*, 54(2), 318–325. http://pspac.info/index.php/dlbh/article/view/340
42. **Lekkala, H. B., & Bandari, V. V. (2026). Deep learning** for weld defect detection using CNN or vision transformers fusion detection. *Power System Protection and Control*, 54(2), 151–158. https://pspac.info/index.php/dlbh/article/view/314
43. **Bandari, V. V., & Lekkala, H. B. (2024). AI-based operator** behavior monitoring and cost optimization using digital traceability in manufacturing systems. *Power System Protection and Control*, 52(1), 38–45. https://pspac.info/index.php/dlbh/article/view/342
44. **Shreyas Jayaprakash,** MACHINE LEARNING-DRIVEN VERIFICATION: USING ML TO OPTIMIZE COVERAGE ANALYSIS, PREDICT SIMULATION RUNTIMES, AND AUTO-GENERATE TESTBENCHES, Vol. 54 No. 2 (2026): April-June 2026, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/391

45. **Mohammed Shafi Kundiladi,** EVENT-DRIVEN IMAGE AND VEHICLE STATUS MANAGEMENT FOR LOW-POWER IOT DIGITAL LICENSE PLATES, Vol. 53 No. 3 (2025): July-September 2025, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/175
DOI: https://doi.org/10.46121/pspc.53.3.17
46. Controlled Synthesis and Characterization of Lanthanum Nanorods, **Author(s), Jayadeep Tejani, Rahul Shah, Hiral Vaghela, Shailesh Vajapara, Amanullakhan Pathan**, doi:10.18576/ijtfst/090205, URL: https://www.naturalspublishing.com/Article.asp?ArtcID=20705, May-2020
47. **Kunal Kapoor**, **Apr 2026,** Relation Extraction and NER through Fine Tuned BERT model with transfer Learning, https://ieeexplore.ieee.org/document/11472214/metrics#metrics
48. **Harender Bisht**, Human–AI Collaboration in Insurance Fraud Detection: Ethical Cloud-Native Architectures for Fair and Transparent Decision Support, **Volume 2026, Issue 1,** https://doi.org/10.52710/cfs.873
49. **Harender Bisht,** Ethical AI-Augmented Compliance Systems: Architectural Innovations for Cloud-Native Financial and Insurance Data Integrity, **Vol. 11 No. 1s (2026) ,** https://doi.org/10.52783/jisem.v11i1s.14056
50. **Harender Bisht**, Governance-By-Design For AI-Based Insurance Fraud Detection: Auditability, Accountability, And Regulatory Traceability, **Archives / Vol. 9 No. 1 (2026**) ,https://doi.org/10.63278/jicrcr.vi.3620
51. **Harender Bisht,** Operationalizing Explainable AI in Insurance Fraud Detection: From Model Transparency to Decision Justification, **Vol. 35 No. 1 (2026): JOCAAA**, https://eudoxuspress.com/index.php/pub/article/view/4791,
52. **A. MahendraVardhan and S. Sridhar,** "Determining False Positive Analysis of Software Vulnerabilities with Predefined Scan Rules using Random Forest Classifier and Decision Tree Technique," **2022 4th International Conference on Advances in Computing**, Communication Control and Networking (ICAC3N), Greater Noida, India, 2022, pp. 622-625, **DOI: https://doi.org/10.1109/ICAC3N56670.2022.10074458**
53. **Amilineni, M.V. 2025**. Audit-Safe Agentic RAG Framework for Regulated Enterprise Systems: A Trustworthy AI Architecture for SAP, Finance, Healthcare, and Government Workflows. INTERNATIONAL JOURNAL OF ADVANCES IN SIGNAL AND IMAGE SCIENCES. (Jan. 2025), 34–46. **DOI: https://doi.org/10.29284/7fjzk883**
54. **Deepa Nagalavi; Shankar Balla Microsoft, USA; Pushpalatha. M; Nashwan Adnan Othman; Malik Bader Alazzam; A. Balakumar, Enhancing Real-Time Language Processing via Advanced PET Signal Analysis and Deep Learning,** ***06-07 June 2025***,
DOI: 10.1109/ICICKE65317.2025.11136561 , https://ieeexplore.ieee.org/document/11136561
55. https://pspac.info/index.php/dlbh/article/view/430
56. **Venkata Sai Aditya Kondru,FMR, Carmel, USA ,; Shankar Balla; Dhavalkumar Thakar; Dheeraj Velaga; Sri Lekha Bandla,** Intelligent Human–Computer Interaction for Navigation Control Through Vision-Based Hand Gesture Recognition, Conferences >2026 IEEE 15th International , **Date of Conference:** ***07-09 April 2026***,**Date Added to IEEE *Xplore*:** 08 May 2026
DOI: 10.1109/CSNT69054.2026.11502317 , https://ieeexplore.ieee.org/abstract/document/11502317
57. **Sudipkumar Ghanvat , Aishwarya Badlani, Shreya Sandeep Joshi,** Marketing Effectiveness in the Post-Cookie Era: A Comparative Analysis of First Party and Zero-Party Data Strategies on Campaign Performance and Customer Equity, **Vol. 31 No. 4 (2023):** JOCAAA , https://www.eudoxuspress.com/index.php/pub/article/view/3940
58. **Navin Chhibber,; Deepak Singh; Anokh Kishore, Capital One, USA; Nikita Chawla; K. Anguraj,** Multi-Granular Attention-Driven Reinforcement Learning Framework for Web Intelligent Enhancement Systems, , **11 June 2026**, https://ieeexplore.ieee.org/document/11483386
59. https://pspac.info/index.php/dlbh/article/view/456

60. https://pspac.info/index.php/dlbh/article/view/455
61. https://eudoxuspress.com/index.php/pub/article/view/5677
62. Anumolu DPK, Veena B, Afreen SS, Bandla J, Lakshmi KC, Pulusu VS. In vitro models for studying drug metabolites and metabolizing enzymes: A comprehensive review. Int J Pharm Investig. 2026;16(3):885–91. Available from: http://dx.doi.org/10.5530/ijpi.20260116
63. Asgar Z, Kumar G, Khandelwal P, Pratap B, Gurjar V, Baluni A, et al. Utility of contrast-enhanced computed tomography abdomen in Intestinal Obstruction. Int J Drug Deliv Technol. 2026;16(32s). Available from: http://dx.doi.org/10.25258/ijddt.16.32s.4
64. Anumolu DP, Ramatulasi J, Ashok G, Afreen SS, Mathew C, Shakar PV. Synthesis and characterization of MIP ghost approach for selective extraction of empagliflozin and quantification by liquid chromatography. J Chromatogr Sci. 2025;63(2). Available from: http://dx.doi.org/10.1093/chromsci/bmaf008.
65. Anumolu DPK, Naraparaju S, Addada SB, Pagidipally V, Pulusu VS, Afreen SS. Synthesis of silver nanoparticles using Ecbolium ligustrinum leaves: Characterization through advanced analytical techniques. Nanotechnol Percept . 2024;710–8. Available from: http://dx.doi.org/10.62441/nano-ntp.vi.3618
66. Naraparaju S, Mukti A, Anumolu DP, Chaganti S. Development and validation of a spectrofluorimetric method for the quantification of capecitabine in bulk and tablets. Turk J Pharm Sci [Internet]. 2023;20(4):234–9. Available from: http://dx.doi.org/10.4274/tjps.galenos.2022.46364
67. Naraparaju S, Yamjala P, Chaganti S, Anumolu DP. Spectrophotometric quantification of atomoxetine hydrochloride based on nucleophilic substitution reaction with 1,2-naphthoquinone-4-sulfonic acid sodium salt (NQS). Turk J Pharm Sci [Internet]. 2024;20(6):405–11. Available from: http://dx.doi.org/10.4274/tjps.galenos.2022.09147.
68. https://pspac.info/index.php/dlbh/article/view/454
69. https://pspac.info/index.php/dlbh/article/view/449
70. https://pspac.info/index.php/dlbh/article/view/453
71. https://eudoxuspress.com/index.php/pub/article/view/5678
72. A Hybrid Deep Learning and Quantum Optimization Framework for Ransomware Response in Healthcare, DOI: 10.1109/OJCS.2025.3648741, https://ieeexplore.ieee.org/document/11316401, Date of Publication: 26 December 2025, Published in: IEEE Open Journal of the Computer Society,Journal: IEEE.
73. Md Nazmul Shakir Rabbi, Farhana Rahman Anonna, Kazi Nehal Hasnain, Sakib Murshed, Md Fazlul Huq Mithu, MD Tushar Khan, Monoara Begum, Sonia Afroze, HYBRID AI MODELS COMBINING RULES AND MACHINE LEARNING FOR FINANCIAL FRAUD CONTROL IN THE UNITED STATES, Vol. 54 No. 2 (2026): April–June 2026, Power System Protection and Control, https://pspac.info/index.php/dlbh/article/view/334 ,https://pspac.info/index.php/dlbh/article/view/334/271
74. **Adegboye A , O., Li, X., Qian, L. (2025).** Parameterized Model for Fork Antenna Design in UWB Band Using Fully Connected Artificial Neural Networks. In: Arabnia, H.R., Deligiannidis, L., Amirian, S., Shenavarmasouleh, F., Ghareh Mohammadi, F., de la Fuente, D. (Eds) Artificial Intelligence and Applications. CSCE 2024. Communications in Computer and Information Science, vol 2252. Springer, Cham. https://doi.org/10.1007/978-3-031-86623-4_7
75. **Adegboye A. Olaoluwa1, Udofia M. Kufre,Obot Akanniyenne**, Inverted C-Shaped Slots Loaded Exponential Tapered Triple Band Notched Ultra-Wideband (UWB) Antenna, https://doi.org/10.5281/zenodo.18139060, **Published May 31, 2024**
76. **Chander Vijay S Sanbhi**, LEADING-INDICATOR-DRIVEN RELIABILITY MODELLING: INCORPORATING OPERATIONS-DRIVEN RELIABILITY TASKS INTO FAILURE HAZARD ESTIMATION, Vol. 54 No. 2 (2026): April-June 2026, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/376.
77. **Chander Vijay S Sanbhi**, DATA QUALITY AS A RELIABILITY MULTIPLIER: QUANTIFYING THE IMPACT OF STANDARDIZED WORK PROCESSES ON RELIABILITY MODEL

ACCURACY, Vol. 52 No. 4 (2024): October-December 2024, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/385

78. **Chander Vijay S Sanbhi,** HYBRID MARKOV–MONTE CARLO RAM MODELLING FOR MULTI-STATE ASSETS WITH NON-EXPONENTIAL FAILURE/REPAIR DISTRIBUTION, Vol. 54 No. 1 (2026): January-March 2026, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/379
79. **Chander Vijay S Sanbhi,** DEGRADATION-AWARE SPARES OPTIMIZATION WITH WEIBULL/PHM UNCERTAINTY AND LOGISTICS DELAYS, Vol. 52 No. 2 (2024): April-June 2024, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/384
80. **Chander Vijay S Sanbhi**, PHYSICS-INFORMED RUL PREDICTION WITH ALEATORIC/EPISTEMIC UNCERTAINTY AND MAINTENANCE SYSTEM INTEGRATION, Vol. 54 No. 1 (2026): January-March 2026, Power System Protection and Control, ISSN-1674-3415 https://pspac.info/index.php/dlbh/article/view/378
81. **Chander Vijay S Sanbhi,** DIGITAL TWIN–RAM CO-SIMULATION FOR MAINTENANCE AND SPARING DECISIONS IN PROCESS FACILITIES, Vol. 52 No. 1 (2024): January-March 2024, Power System Protection and Control, ISSN-1674-3415,https://pspac.info/index.php/dlbh/article/view/383
82. **Chander Vijay S Sanbhi**, A JOINT PROBABILISTIC FRAMEWORK TO INTEGRATE RAM SIMULATION AND ECONOMIC FORECASTING WITHOUT DOUBLE-COUNTING DOWNTIME, Vol. 53 No. 4 (2025): October-December 2025, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/394
83. **Chander Vijay S Sanbhi**, MIXTURE-DISTRIBUTION MAINTAINABILITY MODELLING TO CAPTURE TAIL-RISK DOWNTIME IN CRITICAL MACHINERY, Vol. 53 No. 2 (2025): April-June 2025, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/393
84. **Chander Vijay S Sanbhi,** BAYESIAN UPDATING OF RAM MODELS FOR DYNAMIC AVAILABILITY FORECASTING UNDER REALISTIC DOWNTIME CONSTRAINTS**, Vol. 51 No. 3 (2023): July**-September 2023, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/381
85. **Chander Vijay S Sanbhi,** HYBRID RELIABILITY MODELLING FOR ROTATING EQUIPMENT: PHYSICS-INFORMED LEARNING WITH SPARSE FAILURE DATA, Vol. 51 No. 4 (2023): October-December 2023, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/382
86. https://www.atlantis-press.com/proceedings/icsbpim-25/126017621, **23-24 April 2025**
87. https://ieeexplore.ieee.org/abstract/document/11371052 , **2025 2nd International Conf.**
88. https://ieeexplore.ieee.org/abstract/document/11256577
89. https://ieeexplore.ieee.org/abstract/document/11069968
90. https://ieeexplore.ieee.org/abstract/document/11377462
91. Manikantha Varaprasad Inakollu, (2022, August), Decision Intelligence As A Strategic Capability: A New MIS Framework For AI-Driven Enterprises, A Comprehensive Approach To Integrating Artificial Intelligence Into Organizational Decision-Making, Vol. 23 No. 4 (2022), 66–79, Acta Scientiae, ISSN:2178-7727, URL: https://www.periodicos.ulbra.org/index.php/acta/article/view/662 DOI: https://doi.org/10.22178/acta.23.4.6
92. Manikantha Varaprasad Inakollu, (2022, December), Governance-Centric MIS Architectures For Enterprise-Scale Digital Transformation, Vol. 23 No. 6 (2022), 41-55, Acta Scientiae, ISSN:2178-7727, URL: https://www.periodicos.ulbra.org/index.php/acta/article/view/667 DOI: https://doi.org/10.22178/acta.23.6.4
93. Manikantha Varaprasad Inakollu, (2023, November), Governance-Aware Cloud Architectures For Enterprise Information Systems, Vol. 24 No. 6 (2023), 300-315, Acta Scientiae, ISSN:2178-7727, URL: https://www.periodicos.ulbra.org/index.php/acta/article/view/668

DOI: https://doi.org/10.22178/acta.24.6.28

94. Manikantha Varaprasad Inakollu, (2024, July), Enhancing ERP Auditability and Compliance Using Permissioned Blockchain, A Framework for Transparent and Immutable Enterprise Resource Planning Systems, Vol. 25 No. 3 (2024), 122-137, Acta Scientiae, ISSN:2178-7727,
URL: https://www.periodicos.ulbra.org/index.php/acta/article/view/664
DOI: https://doi.org/10.22178/acta.25.3.16
95. Manikantha Varaprasad Inakollu, (2025, November), Quantum-Resilient Architectures for Enterprise and Cloud Information Systems, Implications of Quantum Computing for Enterprise Cybersecurity and Data Integrity, Vol. 26 No. 3 (2025), 580-595, Acta Scientiae, ISSN:2178-7727,
URL: https://www.periodicos.ulbra.org/index.php/acta/article/view/663
DOI: https://doi.org/10.22178/acta.26.3.44
96. Manikantha Varaprasad Inakollu, (2023, May), Post-Implementation ERP value realization: A Decision intelligence Framework, Vol. 51 No. 2 (2023): April-June 2023, 48-63, Power System Protection and Control, ISSN-1674-3415, URL: https://pspac.info/index.php/dlbh/article/view/286
DOI: https://doi.org/10.46121/pspc.51.2.6
97. Manikantha Varaprasad Inakollu, (2023, July), ERP as Digital Backbone: Redefining enterprise systems for continuous value creation, Vol. 51 No. 3 (2023): July-September 2023, 17-29, Power System Protection and Control, ISSN-1674-3415,
URL: https://pspac.info/index.php/dlbh/article/view/285
DOI: https://doi.org/10.46121/pspc.51.3.4
98. Manikantha Varaprasad Inakollu, (2024, May), FINOPS-Driven Cloud optimization models for enterprise applications, Vol. 52 No. 2 (2024): April-June 2024, 99-110, Power System Protection and Control, ISSN-1674-3415, URL: https://pspac.info/index.php/dlbh/article/view/283
DOI: https://doi.org/10.46121/pspc.52.2.10
99. Manikantha Varaprasad Inakollu, (2025, July), Blockchain-Enabled trust frameworks for enterprise information systems, establishing verifiable trust in distributed organizational environments, Vol. 53 No. 3 (2025): July-September 2025, 285-300, Power System Protection and Control, ISSN-1674-3415, URL: https://pspac.info/index.php/dlbh/article/view/282
DOI: https://doi.org/10.46121/pspc.53.3.19
100. Manikantha Varaprasad Inakollu, (2026, March), Implications of quantum computing for enterprise cybersecurity and data integrity, Vol. 54 No. 1 (2026): January-March 2026, 831-844, Power System Protection and Control, ISSN-1674-3415, URL: https://pspac.info/index.php/dlbh/article/view/281
DOI: https://doi.org/10.46121/pspc.54.1.57
101. **Pawan Kumar Singh,** Scaling Gate-All-Around (GAA) Transistor Architectures for Sub-2nm AI Accelerators Using Atomically Thin 2D Materials, Vol. 33 No. 08 (2024): JOCAAA, Journal Name: Journal of Computational Analysis and Applications, Journal's ISSN: 1521-1398 (Paper),1572-9206 (Online), https://eudoxuspress.com/index.php/pub/article/view/5713
102. **Pawan Kumar Singh,** Energy-Efficient Task Scheduling in Green Cloud Computing Using Neuromorphic Spiking Neural Networks, Vol. 33 No. 08 (2024): JOCAAA, Journal Name: Journal of Computational Analysis and Applications, Journal's ISSN: 1521-1398 (Paper),1572-9206 (Online), https://eudoxuspress.com/index.php/pub/article/view/5712