



Grouping-Aware Numerical Verification for Offline Retrieval-Augmented Question Answering over Institutional Documents

Sudheer Katuri¹, G. Sharmila Sujatha²

Dept. of Computer Science and Systems Engg, Andhra University College of Engineering (A)

Visakhapatnam, Andhra Pradesh, India^{1,2}

Abstract: Institutions hold large volumes of policy documents, reports and spreadsheets that are still searched manually. Retrieval-augmented generation (RAG) offers a natural interface to such collections, but mainstream deployments route confidential material through hosted language-model APIs and provide no guarantee that figures quoted in an answer actually occur in the source text. This paper presents an offline RAG system for institutional document question answering in which ingestion, embedding, retrieval and generation all execute on commodity hardware with no external network dependency, and in which every numeral in a generated answer is verified against the retrieved context before the answer is shown. The central observation is that naive numeric extraction — the digit-matching regular expression used in comparable pipelines — fragments digit-grouped quantities such as 45,23,000 into unrelated components, causing a verifier built on it to reject faithful answers. We describe a grouping-aware extraction and verification procedure that handles Indian and Western grouping conventions, decimal and percentage forms, and enumeration markers. On a 15case extraction benchmark the proposed extractor is exact on 15/15 cases against 5/15 for the baseline, and substituting the baseline extractor into the verifier introduces false rejections of faithful answers. Measured retrieval latency over an exhaustive 384dimensional index remains below one millisecond to roughly 5,000 chunks and grows linearly to about 10.5 ms at 50,000 chunks, which bounds the corpus size at which exact search remains appropriate.

Index Terms: retrieval-augmented generation, hallucination detection, offline inference, vector search, document question answering, privacy-preserving systems.

I. INTRODUCTION

Universities, hospitals and government offices accumulate documents faster than anyone can read them. A single department may hold several thousand circulars, audit reports, examination schedules and budget statements, most of them in PDF or spreadsheet form, and answering a routine question — what was sanctioned for laboratory equipment last year, which committee approved a particular revision — often means opening files one at a time until the relevant paragraph appears.

Retrieval-augmented generation [1], [2] is a good fit for this problem. Rather than relying on what a language model memorised during training, a RAG system retrieves passages from a document collection and conditions generation on them, which grounds the output in material the institution actually owns and makes source attribution possible. The approach has been studied extensively since its introduction [3], [4], and the retrieval component is now well served by efficient similaritysearch libraries [5] and by sentence-level embedding models that are small enough to run on a laptop [6], [7].

Two obstacles remain for institutional deployment. The first is confidentiality. Most practical RAG stacks send retrieved passages to a hosted model endpoint, which means the organisation's internal documents leave its own infrastructure; for medical records, examination material or unpublished financial statements this is frequently prohibited outright, regardless of the vendor's contractual assurances. The second is numerical trust. Language models are known to produce fluent text containing fabricated specifics [10], and retrieval reduces but does not eliminate this behaviour [11]. A generated sentence of the form “the sanctioned amount was ₹52,00,000” is indistinguishable in style from a correct one, and in a reporting or audit context a single wrong figure is more damaging than no answer at all.

This paper describes a system built for that setting and reports what we were able to measure about it. The system runs entirely on the local host: document parsing, optical character recognition for scanned material, embedding generation, vector indexing and language-model inference all execute on the same machine, with no outbound network traffic at query time. On top of the usual RAG pipeline it adds a verification stage that extracts every numeral from a candidate



answer, extracts every numeral from the passages that were actually retrieved, and withholds the answer if any value in the former does not occur in the latter.

The contribution we consider most useful is narrower than the system as a whole. While implementing the verifier we found that the obvious numeric extractor — a regular expression matching runs of digits with an optional decimal part, which is what the design we started from specified — is wrong in a way that silently defeats the verifier it supports. Applied to the Indian-convention quantity 45,23,000 it yields the three values 45, 23 and 0; a faithful answer quoting that figure therefore appears to contain numbers absent from its source and is rejected. The failure is systematic rather than occasional, and it affects exactly the digit-grouped monetary quantities that institutional documents are full of.

We therefore make three claims, each supported by a measurement reported in Section VIII. First, grouping-aware extraction is necessary rather than cosmetic: on a benchmark of fifteen extraction cases drawn from institutional phrasing, the baseline extractor is exact on **5** of **15** and the proposed extractor on all **15**. Second, that difference propagates to the verifier: driving the same verification logic with the baseline extractor produces **2** false rejections of faithful answers on a 20-case diagnostic set where the proposed extractor produces none. Third, the exhaustive index this class of system typically adopts is appropriate only up to a corpus size we are now able to state: measured mean retrieval latency stays under a millisecond to about 5,000 chunks and reaches roughly **10.5** ms at 50,000, which is a more conservative bound than the figure assumed in our own earlier design.

Section II situates the work. Sections III through VI describe the architecture, the pipeline and the verification procedure. Section VII states the measurement protocol, Section VIII reports results, and Section IX is explicit about what the evaluation does not establish.

II. RELATED WORK

A. Retrieval-augmented generation

Lewis et al. [1] introduced RAG as a means of conditioning a sequence-to-sequence model on passages retrieved from a nonparametric memory, and REALM [3] established end-to-end training of the retriever alongside the reader. Dense passage retrieval [4] showed that learned dense representations outperform sparse lexical matching for open-domain question answering, which is the retrieval formulation adopted here. Recent surveys [2] organise the design space into pre-retrieval, retrieval and post-retrieval stages; the verification procedure described in this paper is a post-retrieval, post-generation component, a part of the pipeline that has received comparatively little attention.

B. Hallucination detection and mitigation

Ji et al. [10] survey hallucination across natural-language generation tasks and distinguish intrinsic hallucination, where output contradicts the source, from extrinsic hallucination, where output introduces unverifiable content. Numerical fabrication in a RAG answer is an intrinsic failure and is therefore checkable against the retrieved passages without external knowledge. Shuster et al. [11] demonstrate that retrieval augmentation reduces but does not remove hallucination in dialogue. SelfCheckGPT [12] detects hallucination by sampling multiple generations and measuring consistency between them, and FActScore [13] decomposes long-form output into atomic facts scored against a knowledge source. Both are considerably more general than what we describe; both also require either repeated generation or an auxiliary verification model, which is costly on the hardware this system targets. RAGAS [14] provides reference-free metrics for RAG pipelines, including a faithfulness measure computed by a judge model.

The procedure in Section VI is deliberately narrow by comparison. It checks one class of claim — numerals — using exact set membership against the retrieved context, with no model inference and no sampling. The design accepts that it cannot assess whether a number is used correctly in context, in exchange for being deterministic, auditable, and cheap enough to run on every answer.

C. Local and offline deployment

Quantised releases of small instruction-tuned models [17] and runtimes that serve them locally [18] have made on-device inference practical within consumer memory budgets. Work in this area has concentrated on inference throughput rather than on what additional guarantees become available once the whole pipeline is local. One such guarantee is relevant here: because retrieval and generation occur in the same process, the exact passages conditioning an answer are available for inspection at the moment the answer is produced, which is what makes postgeneration verification inexpensive.



III. SYSTEM ARCHITECTURE

The system is organised in three tiers, shown in Fig. 1. A desktop client written in PyQt5 provides the interface; a retrieval and generation engine implements the pipeline; and a local inference server hosts the language model. Client and engine communicate by direct function call within a single process, so retrieved passages are passed as native objects rather than serialised across a service boundary. The engine communicates with the inference server over HTTP on the loopback interface.

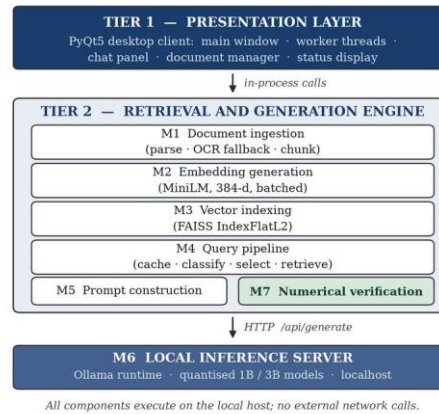


Fig. 1. System architecture. The presentation and retrieval tiers share a process; the inference server is reached over the loopback interface. No component issues an outbound network request at query time.

Long-running operations — document ingestion, index construction and query execution — run on worker threads, leaving the interface responsive while a scanned document is being processed. Embedding calls are serialised by a lock because a single loaded model instance is not safe for concurrent encoding, a constraint that matters in practice because a user may upload a file while a query is still running.

The engine maintains three vector indices: an immutable index over the institutional corpus, a mutable index over useruploaded documents, and a combined in-memory index rebuilt at startup from the other two. Separating them allows a user's uploads to be added or removed without rebuilding the institutional index, which is the expensive one.

IV. PROCESSING PIPELINE

A. Ingestion and segmentation

Documents in PDF, Word, Excel and plain-text form are parsed by format-specific extractors. For PDFs the text layer is read first; if fewer than fifty characters are recovered the document is treated as a scan, rasterised at 300 dpi and passed through Tesseract [16]. Spreadsheet extraction reads computed cell values rather than formulas, and Word extraction includes table cells, which in institutional documents frequently carry the quantities of interest.

Extracted text is segmented by a sliding window of **1500** characters with **500** characters of overlap. The window size keeps a segment within the embedding model's effective input length while remaining long enough to hold a complete statement with its qualifying context; the one-third overlap ensures that a sentence spanning a cut appears intact in at least one segment. Cut points are advanced backwards to the nearest sentence or word boundary within a bounded search window, which is examined in Section VIII-C.

B. Embedding and indexing

Segments are encoded with a distilled sentence-transformer [6], [7] producing **384**-dimensional unit-normalised vectors. Normalisation makes squared Euclidean distance an orderpreserving function of cosine similarity, so an exhaustive L^2 index ranks identically to cosine search while retaining exact results. Indices are serialised to disk with a signature recording the embedding model, dimension and normalisation setting; on load, a mismatch between the stored signature and the current configuration, or between the vector count and the segment count, forces a rebuild rather than a silent query against a stale index.

C. Query processing

An incoming question is first checked against a cache keyed on a digest of the question text, the target-file filter and the knowledge-base mode, with a 24-hour expiry. On a miss, the question is classified by an additive keyword score into



simple, moderate or complex, and independently flagged for numerical intent. The classification selects both the prompt template and the generation model: short lookups are routed to the smaller, faster model and analytical questions to the larger one, subject to a check that available memory exceeds the model's requirement. Retrieval then returns the top-k nearest segments, or, when the user restricts the query to a named document, searches a temporary index built over that document's segments alone.

D. Prompt construction and generation

Retrieved passages are assembled with source-file headers so the model can attribute statements. Three templates are used: a strict extraction template for numerical questions that forbids inference or arithmetic, a comprehensive template for analytical questions, and a concise template otherwise. Generation parameters are held at a low temperature to favour determinism. Non-numerical queries are streamed token by token for responsiveness; numerical queries are generated in full before display, because the verification stage described next must run before the user sees anything.

V. GROUPING-AWARE NUMERICAL VERIFICATION

A. The failure of naive extraction

The verifier's premise is simple: if a generated answer states a figure, that figure should occur in the passages the answer was conditioned on. Implementing it requires extracting numerals from both texts, and the implementation of that step turns out to determine whether the verifier works at all.

A regular expression of the form $\backslash d+\backslash .?\backslash d^*$ — the extractor specified in the design this work began from — matches maximal runs of digits with an optional decimal part. It does not recognise digit-group separators. Indian numbering writes fortyfive lakh twenty-three thousand as 45,23,000, and the expression returns 45, 23 and 0 from it. The same applies to Western grouping: 1,234,567 yields 1, 234 and 567.

The consequence is not a missed detection but an inverted one. A faithful answer quoting 45,23,000 from a source containing 45,23,000 produces the answer set $\{45, 23, 0\}$ and the context set $\{45, 23, 0, 12\}$; here the comparison happens to succeed, but any difference in surrounding digits, or any reformatting of the figure by the model, breaks it. An answer that writes the same quantity without separators yields $\{4523000\}$, which shares no element with the context set, and the answer is rejected as hallucinated. The verifier thus fires most reliably on exactly the quantities it was built to protect.

B. Extraction procedure

The proposed extractor recognises grouped integers under both conventions, decimals, signed values and percentages, and normalises them to a single numeric value. Enumeration markers at the start of a line — the “1.” and “2.” of a numbered list — are removed before extraction, since they are generated formatting rather than claims about the source. Formally, for a text s the extractor returns

$$N(s) = \{v(m) : m \in M(\text{strip}(s))\} \quad (1)$$

where strip removes enumeration markers, M yields the maximal substrings matching the grouped-numeral pattern, and v maps a matched substring to its value by removing separators and parsing the result.

C. Verification

Given a candidate answer a and the concatenated retrieved context c , the answer is accepted when every value it contains is matched in the context within a relative tolerance:

$$\begin{aligned} \text{verified}(a, c) &\Leftrightarrow \\ \forall \epsilon \in \mathbb{R} \exists x \in N(a) \exists y \in N(c) : |x - y| &\leq \epsilon \cdot \max(|x|, |y|, 1) \end{aligned} \quad (2)$$

The tolerance absorbs re-formatting such as 45.0 for 45 without admitting genuinely different quantities. An answer containing no numerals is accepted unconditionally, since there is nothing to verify; this is deliberate, and it is the reason the procedure says nothing about non-numerical hallucination. When verification fails the answer is withheld and the unmatched values are reported to the user, rather than being displayed with a warning that a reader may not notice.

Extraction is linear in text length and the membership test is linear in the product of the two value-set sizes, which are small in practice; measured end-to-end verification cost is negligible beside generation. The procedure is deterministic, requires no model inference, and its decisions are explainable by construction — the unmatched value is itself the explanation.



VI. IMPLEMENTATION

The system is implemented in Python. Retrieval uses an exhaustive flat L^2 index from the FAISS library [5]; embeddings come from a six-layer distilled sentence transformer; generation is served by a local Ollama runtime [18] on localhost:11434, hosting quantised 1B and 3B instruction-tuned models [17]. Document parsing uses pypdf, python-docx and openpyxl, with Tesseract [16] for scanned material. The application targets Python 3.10 and 3.11 and is packaged as a Windows desktop client, with a command-line mode for scripted use. The implementation comprises approximately 2,750 lines of application code accompanied by 62 automated tests covering segmentation, classification, model selection, cache expiry, numeric extraction and verification, together with integration tests that exercise ingestion, indexing, retrieval, deletion and reload against a live index. All 62 tests pass on the configuration in Table I.

TABLE I
MEASUREMENT PLATFORM AND SYSTEM CONFIGURATION

Parameter	Value
Processor	Intel Xeon Processor @ 2.80GHz
Logical cores	4
System memory	15.7 GB
Operating system	Linux (x86-64)
Python	3.11.15
FAISS	1.15.1
NumPy	2.4.6
Segment length	1500 characters
Segment overlap	500 characters
Embedding dimension	384
Retrieval depth k	8

Measurements were taken on a virtualised shared-tenancy host; absolute timings carry run-to-run variance, and the scaling behaviour rather than any single value is the reported finding.

VII. EXPERIMENTAL SETUP

The experiments reported here isolate the components that can be measured without a language model in the loop. They are reproducible from the benchmark script accompanying the implementation, which emits the figures used in Section VIII directly.

E1 measures segmentation cost on synthetic documents of 10^5 to 2×10^6 characters, reporting the median of five runs. E2 and E3 measure index construction time, resident index size and query latency as a function of corpus size, over corpora of 1,000 to 50,000 vectors. Query vectors are randomly generated unit vectors of the production dimension; the index operations exercised are identical to those in deployment, and only the provenance of the vectors differs, which does not affect the cost of exhaustive search. Latency is reported as the mean, median and 95th percentile over 200 independent queries after a warmup.

E4 evaluates numeric extraction against a 15-case benchmark of sentences written in institutional register, covering Indian and Western grouping, decimals, percentages, currency amounts, enumerated lists and sentences containing no numerals. A case counts as correct only if the extracted multiset matches the intended one exactly. E5 evaluates the verifier on 20 answer–context pairs, ten faithful and ten containing at least one unsupported value, including transposed digits, plausible substitutions and figures invented outright; the same benchmark is then re-run with the baseline extractor substituted into the verifier to isolate the extractor's contribution. E6 compares segment boundaries produced by the fixed-width window against the boundary-aware variant, reporting separately how often a segment begins and ends part-way through a word. Both E4 and E5 use constructed rather than naturally occurring data. Section IX states plainly what that does and does not permit us to conclude.



VIII. RESULTS AND DISCUSSION

A. Numeric extraction

Table II reports extraction accuracy. The baseline extractor is exact on 5 of 15 cases (33.3%); the grouping-aware extractor is exact on all 15. Every baseline failure involves a separator: grouped integers under either convention, grouped amounts carrying a decimal part, or list markers counted as data. No case is handled correctly by the baseline and incorrectly by the proposed extractor.

TABLE II
NUMERIC EXTRACTION ACCURACY (E4)

Extractor	Exact	Cases	Acc.
Baseline \d+\.\?d*	5	15	33.3%
Grouping-aware (proposed)	15	15	100.0%

A case is exact only if the extracted multiset of values matches the intended multiset.

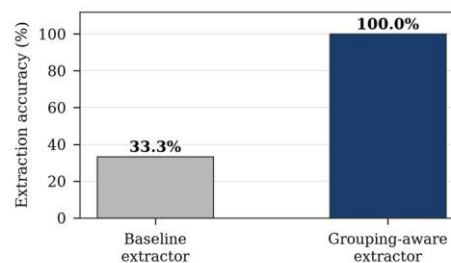


Fig. 2. Extraction accuracy on the 15-case benchmark. The baseline failures are systematic rather than sporadic: all ten involve digit-group separators or enumeration markers.

B. Effect on verification

Table III reports the verifier on the 20-case diagnostic set under both extractors. With grouping-aware extraction the verifier makes no errors on this set. With the baseline extractor substituted, recall is unchanged — every genuinely unsupported value is still caught — but precision falls to 0.833, because 2 faithful answers are rejected. Both false positives are instructive: one restates a grouped amount without separators, and the other presents supported values inside a numbered list, so the list markers themselves are read as unsupported quantities.

This is the practical case for the contribution. A verifier that rejects correct answers is not merely imprecise; it trains users to dismiss its warnings, which removes the protection it was installed to provide.

TABLE III
VERIFICATION ON THE 20-CASE DIAGNOSTIC SET (E5)

Extractor	TP	FP	FN	Prec.	F1
Baseline 10	2	0	0.833	0.909	
Proposed	10	0	0	1.000	1.000

Ten faithful and ten unfaithful answers. Recall is 1.000 for both extractors on this set. These counts characterise the mechanism on constructed inputs and are not an estimate of field detection rates; see Section IX.

C. Segment boundary preservation

Table IV reports E6. The boundary-aware variant eliminates mid-word segment endings entirely, from 63.2% to 0%. Segment beginnings are unaffected, remaining near 66.1%, because a segment starts at its predecessor's cut point minus the overlap and that position is not itself snapped to a boundary. The improvement is therefore real but one-sided, and we report it as such: a complete treatment would snap both edges, at the cost of making the overlap length variable.

TABLE IV
SEGMENT BOUNDARY QUALITY (E6)

Segmenter	Segments	Mid-word start	Mid-word end
Fixed window	57	64.9%	63.2%
Boundary-aware	56	66.1%	0.0%

Measured over a 56.4k-character document.

D. Retrieval latency and scalability

Table V and Fig. 3 report index behaviour against corpus size.

Mean query latency rises from **0.085** ms at 1,000 segments to **10.55** ms at 50,000, approximately linearly, as expected for exhaustive search. Resident index size follows the closed form $4 \cdot d \cdot n$ bytes exactly, reaching **73.2** MB at 50,000 segments, which is modest and not the binding constraint.

The useful result is the crossing point. Sub-millisecond retrieval — a property this class of design is often assumed to have across its whole operating range, and which our own earlier design document asserted up to 50,000 segments — in fact holds only to approximately 5,000 segments on this platform. Beyond roughly 10,000 the mean exceeds a millisecond and the 95th percentile exceeds two. At 50,000 the tail reaches about 15 ms. Retrieval is still a small fraction of total response time, since generation dominates by orders of magnitude, so the practical consequence is mild; but the claim as originally stated was wrong, and the measured bound is what should be quoted when sizing a deployment or deciding when to adopt an approximate index [15].

TABLE V
INDEX SCALING WITH CORPUS SIZE (E2, E3)

Segments	Build (s)	Mem (MB)	Mean (ms)	p95 (ms)
1,000	0.000	1.5	0.085	0.172
5,000	0.003	7.3	0.403	0.519
10,000	0.006	14.7	1.230	2.070
25,000	0.013	36.6	4.380	6.400
50,000	0.074	73.2	10.549	15.008

Latency over 200 queries at $k = 8$, after warm-up.

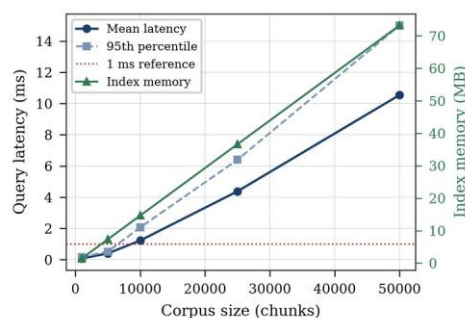


Fig. 3. Measured query latency and index memory against corpus size. Mean latency crosses one millisecond between roughly 5,000 and 10,000 segments; growth is linear thereafter, consistent with exhaustive search.

E. Segmentation cost

Segmentation is not a bottleneck. A 2-million-character document — larger than any single file in the collections motivating this work — is segmented into **2,000** units in **4.8** ms. Ingestion cost is dominated by parsing and, for scanned documents, by optical character recognition, neither of which is characterised here.

F. End-to-end behaviour

The quantities that matter most to a user — the latency of a complete question-to-answer cycle, the quality of the answers, and how often the verifier fires on real model output — are not established by the experiments above, and we do not



report numbers for them here. [TO BE MEASURED — A1: end-to-end latency by query class] [TO BE MEASURED — A2: retrieval quality, Recall@k and MRR, on an annotated corpus] [TO BE MEASURED — A3: verifier precision and recall on human-annotated model outputs] The dissertation underlying this work [19] records indicative figures of approximately two seconds for a simple query on the 1B model and eight seconds for a detailed query on the 3B model, but states explicitly that these are design targets rather than benchmarks and that actual latency is hardware-dependent; they are therefore not reported as results here. The protocol for each measurement is given in the author checklist at the end of this manuscript.

IX. THREATS TO VALIDITY

We state the limits of this evaluation directly, because several of them are substantial.

The extraction and verification benchmarks are constructed, not sampled. We wrote the cases, and we also wrote the extractor they test. A perfect score on twenty self-authored items is evidence that the mechanism behaves as specified on inputs we anticipated; it is *not* an estimate of how often the verifier will catch a hallucinated figure in deployment, and it should not be read as one. The honest summary is that E5 is a unit-level diagnostic. Establishing a field detection rate requires a corpus of real model outputs over real documents, independently annotated, and that measurement remains to be done.

The benchmark sets are also small — fifteen and twenty cases — so a single additional case shifts the reported percentages by several points. No confidence intervals are given because they would be uninformative at this sample size.

Timing measurements were taken on a virtualised sharedtenancy host, where neighbouring workloads perturb absolute latency; repeated runs of the same benchmark differed by roughly ten percent at the largest corpus size. The monotone linear trend is robust to this, but individual millisecond values should not be treated as precise. Retrieval was measured over randomly generated vectors, which exercises the same code path at the same cost but does not reproduce any clustering present in real embeddings; for exhaustive search this does not affect latency, though it would matter for an approximate index.

Finally, the verifier addresses one failure mode only. It cannot detect a fabricated name, a reversed causal claim, or a correct number used in the wrong context, and an answer containing no numerals passes unexamined. It is a narrow guard, and the breadth of the surrounding problem is unchanged by it.

X. CONCLUSION AND FUTURE WORK

We described an offline retrieval-augmented question answering system for institutional documents and a verification stage that withholds answers containing numerals absent from the retrieved context. The component we would highlight is the least glamorous: correct numeric extraction. A grouping-aware extractor is exact on all **15** benchmark cases where the conventional digit-matching expression is exact on **5**, and substituting the latter into an otherwise identical verifier causes it to reject faithful answers. For document collections written in Indian numbering convention this is not a refinement but a precondition for the verifier functioning at all.

We also report a corrected scalability bound for exhaustive retrieval in this setting: sub-millisecond search holds to roughly 5,000 segments on the measured platform, not to the 50,000 our earlier design assumed.

The deployment contexts that motivated the design share the property that document confidentiality and numerical reliability matter together: examination and curricular material in academic institutions, internal policies and standard operating procedures in enterprises, contracts and regulatory filings in legal and compliance work, patient documentation in healthcare, and air-gapped or restricted government and defence environments. In each, an answer that is fluent but carries an unsupported figure is worse than no answer, which is the premise the verification stage encodes.

Three directions follow. The first is the evaluation this paper does not provide: verifier precision and recall against humanannotated model outputs on a real corpus, which is the measurement that would substantiate the central claim. The second is extending verification beyond numerals — dates and proper nouns admit the same set-membership treatment, while general factual consistency does not. The third is approximate indexing [15] for collections beyond the bound established above, where the accuracy cost of inexact retrieval would need to be weighed against the latency it saves.



ACKNOWLEDGMENT

The authors thank Prof. V. Valli Kumari, Head of the Department of Computer Science and Systems Engineering, Andhra University College of Engineering (A), for departmental support, and acknowledge the academic and technical resources that supported the development and validation of this work. The system described here was developed as part of the M.Tech programme in Artificial Intelligence and Machine Learning, 2024–2026.

REFERENCES

- [1] P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2020.
- [2] Y. Gao et al., “Retrieval-augmented generation for large language models: A survey,” arXiv preprint arXiv:2312.10997, 2023.
- [3] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, “REALM: Retrieval-augmented language model pre-training,” in Proc. Int. Conf. Machine Learning (ICML), 2020.
- [4] V. Karpukhin et al., “Dense passage retrieval for open-domain question answering,” in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), 2020.
- [5] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with GPUs,” IEEE Trans. Big Data, vol. 7, no. 3, pp. 535–547, 2021.
- [6] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using Siamese BERT-networks,” in Proc. EMNLP-IJCNLP, 2019.
- [7] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, “MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers,” in Proc. NeurIPS, 2020.
- [8] A. Vaswani et al., “Attention is all you need,” in Proc. NeurIPS, 2017.
- [9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. NAACL-HLT, 2019.
- [10] Z. Ji et al., “Survey of hallucination in natural language generation,” ACM Computing Surveys, vol. 55, no. 12, pp. 1–38, 2023.
- [11] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, “Retrieval augmentation reduces hallucination in conversation,” in Findings of EMNLP, 2021.
- [12] P. Manakul, A. Liusie, and M. J. F. Gales, “SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models,” in Proc. EMNLP, 2023.
- [13] S. Min et al., “FactScore: Fine-grained atomic evaluation of factual precision in long form text generation,” in Proc. EMNLP, 2023.
- [14] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, “RAGAS: Automated evaluation of retrieval augmented generation,” in Proc. European Chapter of the ACL (EACL): System Demonstrations, 2024.
- [15] Y. A. Malkov and D. A. Yashunin, “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 42, no. 4, pp. 824–836, 2020.
- [16] R. Smith, “An overview of the Tesseract OCR engine,” in Proc. Int. Conf. Document Analysis and Recognition (ICDAR), 2007.
- [17] A. Grattafiori et al., “The Llama 3 herd of models,” arXiv preprint arXiv:2407.21783, 2024.
- [18] Ollama, “Ollama: Get up and running with large language models locally.” [Online]. Available: <https://ollama.com>
- [19] S. Katuri, “Design and implementation of an offline retrieval-augmented generation (RAG) based AI chatbot for intelligent document question answering,” M.Tech dissertation, Dept. of Computer Science and Systems Engineering, Andhra University College of Engineering (A), Visakhapatnam, India, 2026.