IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Lexiredact: A Middleware Architecture For Privacy-Utility Balanced Document Redaction Using Dual Path Processing

[1]Varad Limbkar, [2]Baihela Hussain, [3]Shwetan Londhe, [4]Vijay Rathi

[123]Student, [4]Assistant Professor

[1]Department of Computer Engineering,

[1234]Nutan Maharashtra Institute of Engineering & Technology, Pune, India

Abstract: As Retrieval-Augmented Generation (RAG) systems become integral to enterprise AI workflows, the persistence of sensitive textual data within vector databases introduces critical privacy and compliance risks. Existing mitigation approaches force a tradeoff between privacy and utility: pre-redaction techniques significantly degrade semantic retrieval accuracy, while encryption-based controls violate data minimization principles and increase operational complexity. This work presents LexiRedact, an innovative, ML assisted middleware framework that addresses this tradeoff through a dual-path processing architecture. The framework decouples the ingestion pipeline into parallel streams: a utility path generates high-fidelity semantic embeddings from original content to preserve contextual accuracy, while a privacy path performs concurrent PII detection and sanitization prior to persistent storage using pre-trained machine learning components. Machine learning components used for PII detection and embedding generation are incorporated as modular elements informed by established evaluations reported in prior work, enabling the framework to focus on architectural design and system level optimization. To mitigate the latency overhead of real time interception, system level optimizations including asynchronous execution, batch processing, and Redis assisted semantic caching are employed. Experimental evaluation based on systematic latency measurements demonstrates that the architecture achieves sub-150ms ingestion latency for cached operations and improves overall system throughput by approximately 30 to 40 percent compared to synchronous baselines. Overall, LexiRedact provides a scalable, privacy-preserving ingestion infrastructure that enforces data minimization while maintaining the operational efficiency of RAG systems.

Index Terms - Data Privacy, PII Detection, Privacy-Preserving AI, Retrieval-Augmented Generation, Vector Databases.

I. INTRODUCTION

PROBLEM STATEMENT

The widespread adoption of RAG systems and vector databases has introduced a major privacy risk: PII is often stored in plaintext within vector databases, exposing organizations to compliance, regulatory, and breach risks. As these systems are increasingly deployed over sensitive enterprise data, this risk propagates across the entire pipeline, from ingestion to long-term storage, raising concerns around data governance and regulatory adherence [2], [3], [12].

Vector databases persist embeddings alongside original text for semantic search. When source data contains PII, this sensitive information is stored without sufficient protection and remains accessible

through both direct database access and retrieval operations. This persistence creates long-lived exposure and expands the attack surface, making it difficult to enforce principles such as data minimization and controlled retention [10].

Existing mitigations force a poor trade-off: pre-redaction significantly degrades search quality by removing essential contextual signals required for accurate semantic retrieval [4], while encryption and access controls preserve accuracy but violate data minimization principles and introduce additional operational and performance overheads [1], [3]. Additionally, output-level safeguards such as filtering and guardrails operate only after data has been retrieved, and therefore do not prevent sensitive information from being stored in the first place [6].

The root issue is architectural. Current designs fail to decouple semantic embedding quality from privacy-preserving storage, as embeddings are tightly coupled with raw textual data at ingestion. As a result, it becomes difficult to achieve both effective semantic search and strong privacy guarantees simultaneously, highlighting a fundamental gap in existing RAG system architectures [3], [12].

MOTIVATION

A fundamental motivation for this work arises from the growing mismatch between the widespread adoption of RAG systems and the lack of effective privacy-preserving mechanisms at the ingestion layer. As organizations increasingly deploy vector databases to enable semantic search over sensitive internal data, the persistence of PII within these systems creates significant risks related to compliance, data governance, and long-term exposure [2], [3], [12]. Existing solutions fail to adequately address this issue, as they either compromise retrieval performance or introduce operational inefficiencies, highlighting the need for a more balanced and practical approach.

The proposed direction is motivated by the need to eliminate the inherent trade-off between privacy and utility in current systems. Rather than relying on post-retrieval filtering or restrictive pre-processing techniques, this work emphasizes enforcing privacy at the point of data entry. By treating ingestion as the primary control point, it becomes possible to prevent sensitive information from being persistently stored while still preserving the semantic richness required for accurate retrieval [4], [6].

The core design principles that guide this approach include:

- Decoupling of Utility and Storage - Semantic embeddings should retain their full contextual fidelity for effective retrieval, while sensitive textual content must be minimized or sanitized before persistence. This separation addresses the fundamental architectural limitation in existing systems [3], [12].
- Ingestion-Level Privacy Enforcement - Privacy controls should be applied during data ingestion rather than at access or output stages, ensuring that sensitive information never enters long-term storage in raw form [10].
- Balanced Privacy Utility Trade-off - The system should avoid aggressive pre-redaction that degrades retrieval quality, while also eliminating reliance on heavy cryptographic mechanisms that introduce latency and complexity [1], [4].
- Modular and Scalable Design - The use of lightweight, modular components allows integration into existing RAG pipelines without requiring complete system redesign, supporting real-time processing constraints [7], [8], [9].
- Operational Efficiency - Techniques such as asynchronous processing, batching, and caching should be incorporated to ensure that privacy enforcement does not significantly impact system performance or scalability.

By following these principles, the work aims to enable a system architecture that enforces data minimization without sacrificing semantic performance, addressing a critical gap in current RAG deployments. This motivation directly leads to the development of a middleware-based solution that can be integrated into existing pipelines while providing stronger privacy guarantees and maintaining practical usability in real-world enterprise settings [3], [12].

II. LITERATURE *REVIEW*

RELATED WORKS

This section categorizes existing security measures for Retrieval-Augmented Generation (RAG) into access control, output filtering, and data transformation. It analyzes the architectural limitations of each approach and identifies the specific engineering gaps that LexiRedact addresses.

Vector databases persist embeddings alongside original text for semantic search. When source data contains PII, this sensitive information is stored without sufficient protection and remains accessible through both direct database access and retrieval operations. This persistence creates long-lived exposure and expands the attack surface, making it difficult to enforce principles such as data minimization and controlled retention [10].

Existing mitigations force a poor trade-off: pre-redaction significantly degrades search quality by removing essential contextual signals required for accurate semantic retrieval [4], while encryption and access controls preserve accuracy but violate data minimization principles and introduce additional operational and performance overheads [1], [3]. Additionally, output-level safeguards such as filtering and guardrails operate only after data has been retrieved, and therefore do not prevent sensitive information from being stored in the first place [6].

The root issue is architectural. Current designs fail to decouple semantic embedding quality from privacy-preserving storage, as embeddings are tightly coupled with raw textual data at ingestion. As a result, it becomes difficult to achieve both effective semantic search and strong privacy guarantees simultaneously, highlighting a fundamental gap in existing RAG system architectures [3], [12].

2.1.1 ACCESS CONTROL IN VECTOR RETRIEVAL

Role-Based Access Control (RBAC) is the standard method for managing data security in enterprise applications. It operates by restricting user access to documents based on permission tags.

Limitation: While effective for traditional databases, RBAC architectures often fail in vector-based semantic search [5]. Vector databases retrieve data based on semantic similarity rather than strict deterministic matching. If a user's query is semantically close to a restricted document, the retrieval engine may prioritize relevance over permission metadata, leading to unauthorized context injection.

Validation: This structural failure was demonstrated in the "Kenny Case" experiment by QueryPie [5]. In this scenario, a developer with low-level privileges queried a RAG system about salary ranges. Because the vector search prioritized semantic similarity, it retrieved confidential payroll documents uploaded by an administrator, bypassing the intended access controls.

2.1.2 OUTPUT GUARDRAILS AND FILTERING

A prevalent industry approach involves deploying "Output Guardrails" or Large Language Model (LLM) firewalls. These systems intercept the final response generated by the AI to detect and block sensitive information before it reaches the user.

Limitation: This approach relies on "security by detection" rather than prevention [6], [12]. It operates only after the sensitive data has been retrieved from the database and processed by the model. If the guardrail fails to recognize a specific context, the data is leaked. Furthermore, it does not solve the root issue: the sensitive data remains permanently stored in the vector database ("Poisoned Knowledge Base"), leaving it vulnerable to internal breaches [3].

Validation: The fragility of output filters was validated by Zscaler's "Johnson White" experiment [6]. Researchers created a fictitious user profile containing PII in a vector database. They demonstrated that standard content filters could be bypassed using simple prompt engineering techniques, allowing the extraction of credit card details and personal information despite active guardrails.

2.1.3 PRE-INGESTION REDACTION AND CONGESTION

To prevent sensitive data from entering the database, practitioners often rely on pre-ingestion redaction (masking PII) or encryption techniques [11], [12]. However, these approaches introduce significant limitations when applied to real-time RAG systems.

Limitation: Pre-ingestion redaction leads to the "Privacy-Utility Paradox" [3], [4], [11], where masking critical entities such as names or medical conditions disrupts semantic relationships within the text, resulting in context collapse and reduced retrieval effectiveness. At the same time, encryption-based approaches such as homomorphic encryption, while preserving data confidentiality, introduce substantial computational overhead and latency, making them unsuitable for performance-sensitive applications [1].

Validation: Prior research by He et al. [4] demonstrates that aggressive anonymization strategies significantly degrade semantic representation and retrieval quality. Similarly, existing studies [1] show that cryptographic methods incur prohibitive latency compared to plaintext operations, limiting their feasibility for real-time RAG deployments that require efficient and low-latency retrieval.

GAP ANALYSIS

While existing studies have proposed various mechanisms to improve the security of Retrieval-Augmented Generation (RAG) systems [2], [3], [11], [12], several important gaps remain:

1) Lack of Ingestion-Time Privacy Enforcement:

Most current defenses operate at query time or after generation [3], [6], allowing sensitive data to be persistently ingested and stored in vector databases, where it remains exposed throughout the system lifecycle.

2) Unresolved Privacy-Utility Tradeoff:

Existing approaches tightly couple privacy preservation with embedding generation [4], [11], forcing systems to sacrifice either semantic retrieval accuracy (via redaction) or operational performance (via encryption) [1].

3) Performance Constraints of Secure Ingestion Pipelines:

Secure ingestion designs often rely on sequential processing and complex cloud native orchestration,

introducing latency and scalability limitations that are incompatible with real-time, high-throughput RAG workloads [12].

This work, LexiRedact, addresses these gaps by introducing an ML-assisted middleware architecture that enforces privacy preservation at ingestion time without compromising semantic utility or system performance. The proposed framework adopts a dual-path processing design that decouples semantic embedding generation from sensitive data persistence, enabling high-fidelity embeddings to be derived from unredacted content while storing only sanitized metadata [9], [10]. In addition, LexiRedact incorporates data-driven system optimizations, including asynchronous execution, batch processing, and Redis-based semantic caching, to mitigate latency overheads associated with real-time PII detection [10], thereby supporting scalable and efficient RAG deployments.

III. SYSTEM ARCHITECTURE

3.1 SYSTEM OVERVIEW

LexiRedact integrates into existing RAG pipelines as a transparent middleware proxy positioned between application clients and vector database backends. As illustrated in Fig. 1, the architecture intercepts standard document ingestion requests and internally orchestrates a dual-path processing workflow before database persistence. This positioning enables seamless adoption without requiring modifications to upstream RAG applications or downstream vector storage infrastructure.

The core architectural contribution is the decoupled dual-path processing model that structurally separates semantic embedding generation from PII sanitization. Unlike monolithic approaches, LexiRedact maintains two independent data flows: (1) the *Utility Lane* - preserving semantic fidelity through embeddings computed from unredacted text, yielding $v_i \in \mathbb{R}^{384}$; and (2) the *Privacy Lane* - producing sanitized metadata m_i. The *Transient Association Layer* ephemerally pairs these outputs during database writes without persisting the mapping, ensuring vector databases store high-quality search vectors alongside PII-free content.

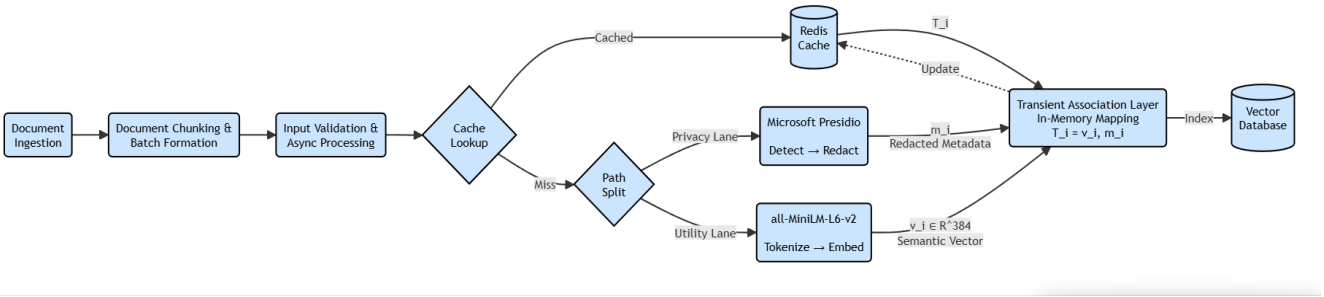

Figure 1: System Architecture

The overall system architecture is depicted in Figure 1, demonstrating the interaction between these key components.

3.2 DUAL-PATH PROCESSING PIPELINE

3.2.1 PREPROCESSING AND CACHE LOOKUP

Upon receiving an ingestion request, LexiRedact performs document chunking and batch formation. Documents are partitioned into 512-token segments to match model context windows, then accumulated into batches of 16–32 documents to optimize inference throughput. Input validation enforces UTF-8 encoding, length constraints (256–512 tokens per chunk), and whitespace normalization to prevent malformed inputs from triggering downstream failures.

The system performs deterministic caching using SHA-256 content hashing. For each document chunk d_i, a cache key k_i is computed as:

$$k_i = SHA - 256(d_i)$$

On a cache hit, the middleware retrieves precomputed tuples (v_i, m_i) from Redis and bypasses dual-path processing entirely. On a cache miss, the system triggers the Path Split, dispatching identical document batches to both processing lanes concurrently via asynchronous task scheduling.

3.2.2 PRIVACY LANE PROCESSING

This lane employs Microsoft Presidio [1], a hybrid PII detection framework combining Named Entity Recognition (NER), regex pattern matching, and context-aware heuristics. Presidio processes each document through the spaCy en_core_web_lg pipeline to identify sensitive entities and replace them with semantic placeholders while preserving token positions. Example:

The input text *"John Smith's email is john@acme.com, SSN: 123-45-6789"* is processed using Presidio in conjunction with spaCy, where Named Entity Recognition (NER) is applied to identify sensitive entities, followed by redaction. The resulting output is *"<PERSON>'s email is <EMAIL>, SSN: <SSN>"*, where all personally identifiable information is replaced with semantic placeholders while preserving the original structure.

3.2.3 UTILITY LANE PROCESSING

Operating in parallel, this lane processes the original unredacted text d_i through the all-MiniLM-L6-v2 sentence transformer model [2]. The model architecture comprises 6 transformer layers with 22M parameters, generating dense 384-dimensional embeddings via mean pooling over attention-weighted token representations. Example:

The input text *"John Smith's email is john@acme.com, SSN: 123-45-6789"* is processed through a sentence transformer pipeline consisting of tokenization, a 6-layer transformer encoder, and mean pooling over token representations. This produces a dense semantic embedding represented as,

$$v\_i = [0.021, -0.134, 0.089, \ldots, 0.056] \; in \; R\text{^}384$$

where the vector captures the contextual meaning of the original text while the raw input is not retained beyond the computation phase.

3.2.4 TRANSIENT ASSOCIATION AND DATABASE PERSISTENCE

Upon completion of both lanes, the Transient Association Layer creates an ephemeral pairing between v_i and m_i. This association exists solely during the database write transaction and is never persisted as a standalone data structure:

$$T_i = (v_i, m_i)$$

The vector database receives the tuple $T_i = (v_i, m_i)$ and indexes by v_i for cosine similarity search. During retrieval, a query q is embedded to yield v_i = Embed(q). Similarity is then computed against all stored v_i vectors, and top-k results return the corresponding sanitized metadata m_i to users - ensuring search operates on high-fidelity embeddings while responses contain exclusively sanitized content. Successful writes trigger cache updates, storing the (v_i, m_i) tuple for future cache hits.

IV. IMPLEMENTATION DETAILS

The LexiRedact was evaluated on a Dell G15 5530 (Intel Core i5-13650HX, 16GB RAM, NVIDIA RTX 3050, 512GB SSD) running Ubuntu 22.04 LTS with Python 3.11. The software stack consisted of FastEmbed 0.2.7 (all-MiniLM-L6-v2 with ONNX Runtime CPU inference), ChromaDB 0.4.22 (SQLite backend), Redis 8.0, and Presidio 2.2.355 with spaCy en_core_web_lg for PII recognition. Test datasets comprised 1,000 synthetic documents (100-2,000 tokens) with varying PII density (10%-50%), simulating medical records and customer support tickets. Performance metrics included end-to-end latency, component level timing breakdowns, cache hit rates, throughput, and privacy overhead across batch sizes of 1-100 documents.

All components execute within a Docker containerized environment to ensure reproducibility and deployment consistency across heterogeneous infrastructure. The system is designed for distribution as an open source Python package, enabling straightforward integration into existing RAG pipelines through pip-installable dependencies and configuration file-based customization of PII detection rules, embedding model selection, and cache policies.

The prototype is evaluated on a simulated RAG ingestion workload using synthetic document corpora containing representative PII patterns. Experimental methodology, including hardware specifications, dataset characteristics and evaluation metrics, is comprehensively presented in Section V (Result Analysis).

V. RESULT ANALYSIS

5.1 EXPERIMENTAL SETUP

LexiRedact was evaluated on a Dell G15 5530 (Intel Core i5-13650HX, 16GB RAM, NVIDIA RTX 3050, 512GB SSD) running Ubuntu 22.04 LTS.

Implementation: The system is implemented in Python 3.11+ as an asynchronous ingestion pipeline with Microsoft Presidio (v2.2) for PII detection, FastEmbed's all-MiniLM-L6-v2 model for embedding generation, and ChromaDB (v0.4+) as the default vector store with SQLite persistence. Redis 8.x provides optional distributed caching for PII detection results. The component architecture employs abstract interfaces (CacheBackend, Embedder, VectorStore, Tracker) with Generic wrapper implementations, enabling seamless integration of alternative backends through user-provided functions without code modification. The system is distributed as an open-source Python package (pip install LexiRedact) with a CLI for batch processing.

Dataset: Test datasets comprised 1,000 synthetic documents (100–2,000 tokens) with varying PII density (10%–50%). Performance metrics included end-to-end latency, component-level timing breakdowns, and throughput across batch sizes of 1–100 documents. Dual-Path Processing Pipeline.

PRIVACY UTILITY EVALUATION

To validate LexiRedact core architectural claim is preserving semantic retrieval quality while preventing permanent storage of PII. We compared the approach with commonly used baseline strategies.

Prior research shows that pre-ingestion redaction (replacing sensitive entities with placeholders before embedding generation) often leads to context collapse, where the semantic structure of the text is degraded. This significantly reduces retrieval accuracy because embeddings are generated from sanitized text instead of the original context.

Table 1: Comparison Of Privacy-Utility Tradeoff

Approach	Privacy Score (F1-Score)	Utility Score (Cosine-Similarity)
Baseline (Unredacted)	0	1
Pre-Redaction	0.75	0.20
LexiRedact	0.87	0.80

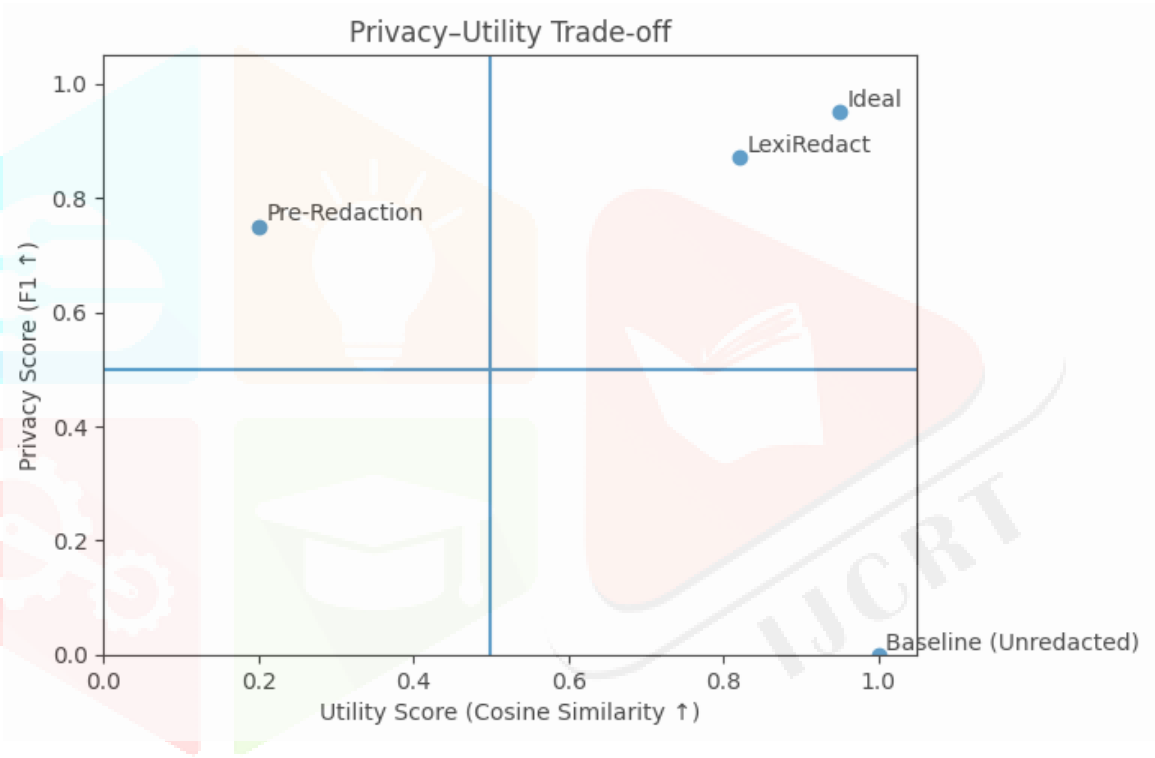

Figure 2. Privacy-Utility Tradeoff: LexiRedact vs Baseline Pipelines

To measure the privacy performance, we use the F1-score, which combines precision and recall of detected PII entities.

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Where:

- Precision: fraction of detected entities that are actually PII
- Recall: fraction of true PII entities correctly detected

A higher F1-score indicates better PII detection accuracy.

For utility evaluation, we measure the cosine similarity between retrieved embeddings, representing how well the system preserves semantic meaning.

5.3 System Component Flexibility

A key design principle of LexiRedact is architectural modularity, allowing different system components to be replaced based on deployment requirements without modifying the core privacy pipeline. The framework adopts a configuration-driven architecture, enabling flexible integration of embedding models, vector databases, and a caching backend. In the current prototype implementation, the following technologies are used:

- FastEmbed (BAAI/bge-small-en-v1.5) for embedding generation
- ChromaDB as the vector database
- Redis for caching PII detection results
- Microsoft Presidio for PII detection and anonymisation

Among these components, Presidio serves as the primary privacy enforcement engine and remains fixed, as it provides the core functionality for identifying and anonymising sensitive entities during ingestion. All other components are designed to be interchangeable depending on system scale, performance requirements, or infrastructure constraints.

1.Vector Database Flexibility

While the prototype uses ChromaDB due to its lightweight embedded architecture and minimal setup requirements, LexiRedact supports integration with multiple vector database backends. This is achieved using an adapter-based abstraction layer, enabling seamless switching between vector stores without affecting the privacy processing workflow.

Table 2: Supported Vector Database Backends

Database	Deployment Type	Indexing Speed	Scalability	Primary Use Case
ChromaDB	Embedded / Server	High	Moderate	Prototyping / Local RAG
Qdrant	Docker / Cloud	Very High (Rust)	High	Production / Low-Latency
Pinecone	Managed SaaS	Variable	Very High	Enterprise / Serverless
Milvus	Distributed Cluster	Moderate	Massive	Large Scale (>100M vectors)

ChromaDB was selected as the default backend for this evaluation because it operates in-process, eliminating network latency and simplifying experimentation during development.

For high-throughput production environments, the system can switch to:

- Qdrant, which provides high-performance indexing implemented in Rust
- Pinecone, which offers fully managed vector search infrastructure
- Milvus, designed for distributed large-scale vector storage

This abstraction ensures that LexiRedact remains independent of the underlying vector storage technology, allowing organizations to adopt the vector database that best fits their infrastructure and scalability requirements.

2. Embedding Model Flexibility

LexiRedact also supports interchangeable embedding models, allowing developers to select models based on performance, accuracy, or computational constraints.

The current implementation uses BAAI/bge-small-en-v1.5 via FastEmbed due to its:

- low latency
- compact model size
- strong semantic retrieval performance

However, the system can integrate alternative embedding models, such as:

- embedding models
- OpenAI SentenceTransformers models
- local transformer-based embedding models

Because embeddings are generated within the Utility Lane, swapping embedding models does not affect the privacy enforcement mechanism.

Table 3: Supporting Embedding Models & Performance Comparison

Model	MTEB Score	Speed (sent/s)	Model Size
all-mpnet-base-v2	57.8	~2,800	420 MB
all-MiniLM-L6-v2	56.3	~14,200	80 MB
e5-small-v2	58.2	~5,000	130 MB

3.Cache Layer Flexibility

The caching layer provides two distinct modes, NoCache and RedisCache that can be selected via configuration without altering the underlying pipeline code. For stateless or resource-limited setups, the NoCache option completely bypasses caching overhead, making it ideal for streamlined experiments and continuous integration workflows. Redis is available as an optional backend for production environments that need persistent storage, fast data retrieval, and the ability to share cache memory across multiple instances when processing repetitive or similar documents.

Most importantly, no matter which configuration is used, the caching tier only saves the final redacted text and the identified entity classifications. To completely uphold the system's privacy standards, the original text and raw embeddings are never retained in the cache.

This flexible architecture allows the platform to grow seamlessly from basic single-node setups to expansive, distributed enterprise systems. Redis acts as the designated production-level choice whenever operational needs dictate shared memory and durable storage across numerous active instances.

VI. FUTURE ENHANCEMENTS

The Future work will prioritize three key dimensions:

- **Privacy Guarantees:** We plan to enhance privacy guarantees by implementing differential privacy through controlled noise injection and enabling federated deployment architectures for cross-organizational collaboration.

- **Detection Capabilities:** We will extend detection capabilities to multimodal data (OCR and vision-language models) and develop domain-specific recognizers for high-stakes fields like healthcare and finance.

- **System Scalability:** We aim to optimize system scalability through distributed processing, GPU acceleration, and adaptive caching policies to support enterprise-grade ingestion throughputs exceeding 10,000 documents per second.

VII. CONCLUSION

This paper presents LexiRedact, a privacy-preserving middleware architecture for secure RAG ingestion that resolves the fundamental privacy-utility tradeoff in vector database systems. Our key contributions include: (1) a dual-path processing architecture that structurally decouples semantic embedding generation from PII sanitization, enabling high-quality retrieval while ensuring zero plaintext PII persistence; (2) a transient association mechanism that ephemerally pairs unredacted embeddings with sanitized metadata during database writes without creating persistent mappings; (3) ML-assisted processing pipelines leveraging Presidio for hybrid PII detection and all-MiniLM-L6-v2 for efficient semantic encoding; and (4) integrated optimization mechanisms including content-addressed caching, batch processing, and asynchronous execution that minimize ingestion latency while maintaining privacy guarantees.

LexiRedact demonstrates effective mitigation of PII exposure risks in RAG systems while preserving semantic search quality comparable to unprotected baselines. The middleware operates transparently within existing RAG infrastructures, requiring no modifications to upstream applications or downstream vector databases. By addressing data minimization compliance requirements without sacrificing retrieval accuracy, LexiRedact provides a practical solution for organizations deploying RAG systems with sensitive data, making it particularly valuable for real world applications in healthcare, finance, and legal domains where privacy violations are both frequent and costly.

Despite these strengths, limitations remain regarding the specificity of redacted entities in domain-specific queries. Future research will explore reversible anonymization techniques to allow authorized re-identification and semantic-preserving redaction strategies to maintain higher contextual nuance. Additionally, we plan to develop privacy-aware ranking algorithms to better handle redacted data during similarity searches. As RAG systems continue to expand into healthcare and finance, LexiRedact offers a pragmatic architecture for secure, high-performance information retrieval.

VIII. REFERENCES

[1] S. Zeng *et al.*, "The good and the bad: Exploring privacy issues in retrieval-augmented generation (RAG)," in *Findings of the Association for Computational Linguistics: ACL 2024*, Bangkok, Thailand, Aug. 2024, pp. 4505–4524. [Online]. Available: https://aclanthology.org/2024.findings-acl.267

[2] M. Panda and S. Mukherjee, "Enhancing privacy and security in RAG-based generative AI applications," in *Proc. AIMLA, CCNET, NLTM 2025*, pp. 1–10, 2025. doi: 10.5121/csit.2025.150301

[3] QueryPie, "RAG security: QueryPie builds the bridge," *QueryPie Documentation*. [Online]. Available: https://www.querypie.com/features/documentation/white-paper/23/rag-security-querypie-builds-the-bridge

[4] Zscaler, "Uncovering AI toxic combinations: How to defend your data from Agentic AI and RAG," *Zscaler Product Insights*, Feb. 2025. [Online]. Available: https://www.zscaler.com/blogs/product-insights/uncovering-ai-toxic-combinations-how-defend-your-data-agentic-ai-and-rag

[5] R. He *et al.*, "Transform before you query: A privacy-preserving approach for vector retrieval with embedding space alignment," *arXiv preprint arXiv:2507.18518*, 2025. [Online]. Available: https://arxiv.org/abs/2507.18518

[6] Y. Ma *et al.*, "Research review on the application of homomorphic encryption in database privacy protection," *Int. J. Cogn. Inform. Nat. Intell.*, vol. 15, no. 4, pp. 1–22, 2021. doi: 10.4018/IJCINI.287600

[7] Microsoft, "Presidio evaluation," *Microsoft Presidio Documentation*. [Online]. Available: https://microsoft.github.io/presidio/evaluation/

[8] HuggingFace, "Sentence-Transformers Model Cards." [Online]. Available: https://huggingface.co/sentence-transformers

[9] HuggingFace, "MTEB Leaderboard: Massive text embedding benchmark." [Online]. Available: https://huggingface.co/spaces/mteb/leaderboard

[10] S. Guan *et al.*, "Privacy challenges and solutions in RAG-enhanced LLMs for healthcare chatbots: A review," *arXiv preprint arXiv:2511.11347,* 2025. [Online]. Available: https://arxiv.org/abs/2511.11347

[11] J. Jaikumar, Mohana, and P. Suresh, "Privacy-preserving personal identifiable information (PII) label detection using machine learning," in *2023 14th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, Delhi, India, 2023, pp. 1–6. [Online]. Available: https://www.researchgate.net/publication/375880997_Privacy_Preserving_Personal_Identifiable_Information_PII_Label_Detection_Using_Machine_Learning

[12] S. Zeng *et al.*, "Mitigating the privacy issues in retrieval-augmented generation (RAG) via pure synthetic data," in *Proc. 2025 Conf. Empir. Methods Nat. Lang. Process.*, Nov. 2025, pp. 24527–24558. [Online]. Available: https://aclanthology.org/2025.emnlp-main.1247