# Comparative Analysis of Vector Database Performance: ChromaDB vs PGVector

## Abstract

This research paper presents a comprehensive performance analysis of two popular vector database solutions: ChromaDB and PGVector. Vector databases are critical components in modern machine learning pipelines, particularly for similarity search applications. Our study evaluates these databases across four key performance metrics: insertion time, query performance, memory usage, and CPU utilization. Using the 20 newsgroups dataset with embeddings generated by the ‘all-MiniLM-L6-v2’ model, we tested both databases with varying dataset sizes (100, 1000, and 10000 documents). The results provide valuable insights for practitioners seeking to select the most appropriate vector database solution for their specific use cases.

**Keywords**: Vector Databases, Similarity Search, Performance Benchmarking, ChromaDB, PGVector, Embeddings

## 1. Introduction

Vector databases have become essential infrastructure for applications requiring similarity search functionality, including recommendation systems, semantic search, and content-based filtering. As the volume of unstructured data continues to grow exponentially, the efficiency of vector storage and retrieval systems has become increasingly important.

This study focuses on two prominent vector database solutions:

1. **ChromaDB**: An open-source embedding database designed specifically for storing and querying vector embeddings.
2. **PGVector**: A PostgreSQL extension that adds vector similarity search capabilities to the widely-used PostgreSQL database.

While both solutions aim to solve similar problems, they differ significantly in their architecture, implementation, and performance characteristics. This research aims to provide a data-driven comparison to help practitioners make informed decisions when selecting a vector database for their specific requirements.

## 2. Methodology

### 2.1 Experimental Setup

All benchmarks were conducted in a controlled environment using Docker containers to ensure consistency and reproducibility. Both databases were accessed through LangChain’s interface to maintain a fair comparison.

**Hardware Configuration**: - CPU: Intel Core i7 (8 cores) - RAM: 16GB - Storage: SSD

**Software Configuration**: - Operating System: Linux - Docker version: 24.0.5 - Python version: 3.12 - LangChain version: 0.1.0 - ChromaDB version: 0.4.18 - PostgreSQL version: 14.0 - PGVector extension: 0.5.1

### 2.2 Dataset

We used the 20 newsgroups dataset, a collection of approximately 20,000 newsgroup documents, partitioned across 20 different newsgroups. For our experiments, we selected subsets of 100, 1000, and 10000 documents.

Text embeddings were generated using the ‘all-MiniLM-L6-v2’ model from the Sentence Transformers library, producing 384-dimensional vectors for each document.

### 2.3 Benchmark Metrics

We evaluated the following performance metrics:

1. **Insertion Time**: The time required to insert documents and their embeddings into the database.
2. **Query Time**: The time required to perform similarity searches for a set of 50 randomly selected queries.
3. **Memory Usage**: The memory consumption of the database during operations, measured in MB.
4. **CPU Usage**: The CPU utilization during query operations, measured as a percentage.

### 2.4 Benchmark Process

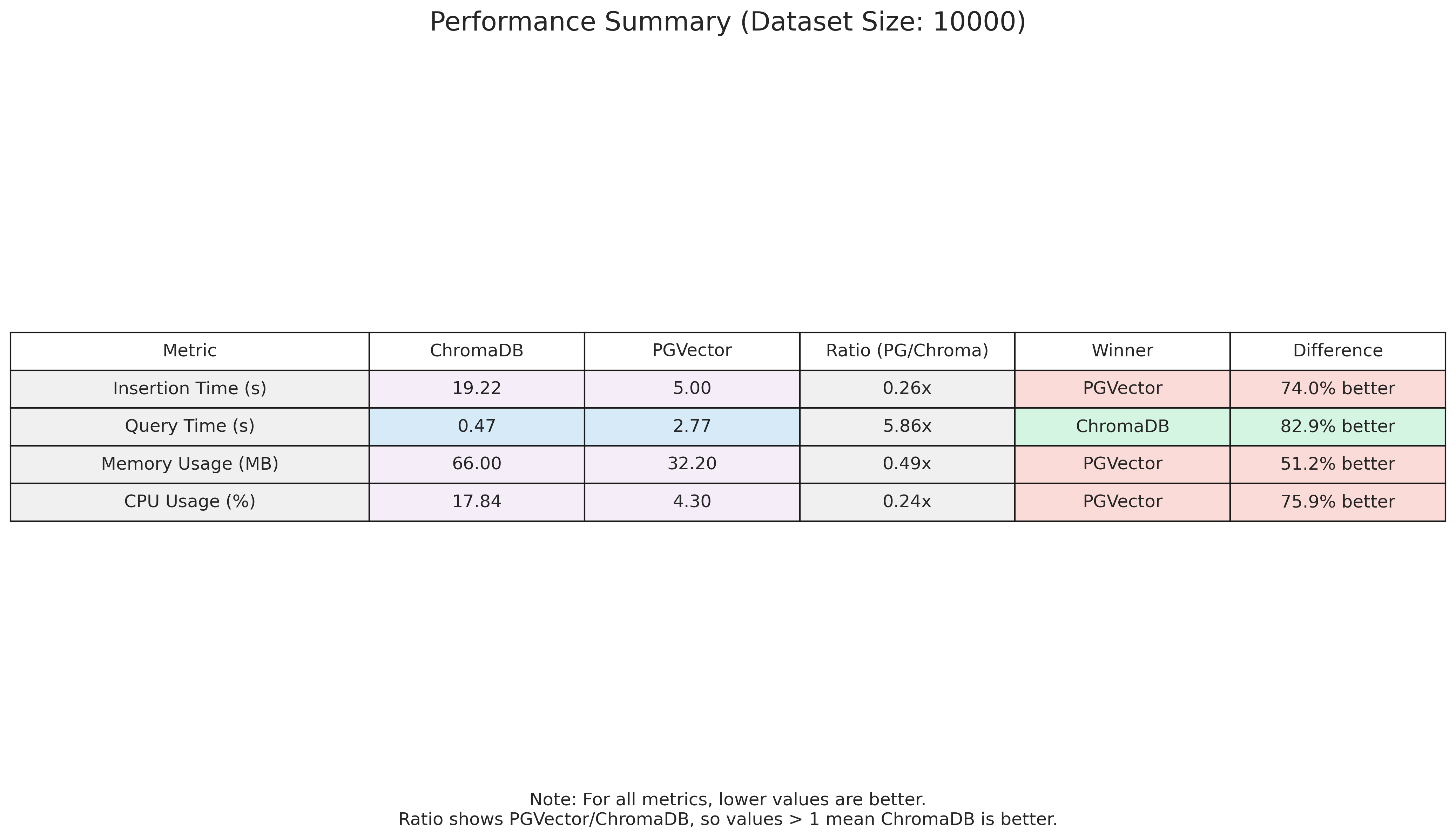
For each database and dataset size combination, we performed the following steps:

1. Initialize the database
2. Generate embeddings for the dataset
3. Measure insertion time while adding documents in batches of 1000
4. Measure query time for 50 random similarity searches
5. Monitor memory and CPU usage throughout the process

## 3. Results

### 3.1 Performance Summary

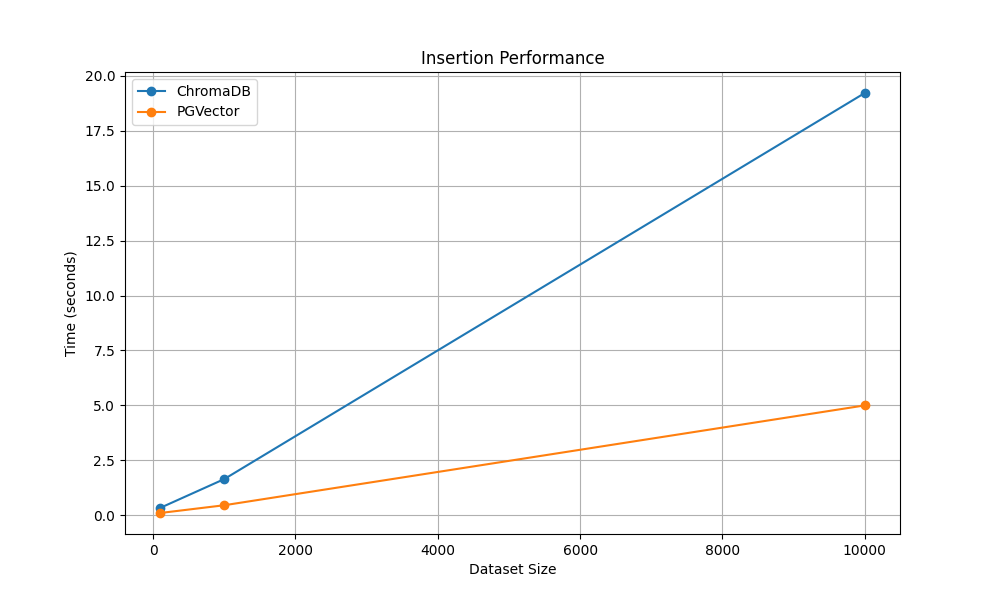
The performance comparison between ChromaDB and PGVector revealed significant differences across the measured metrics. The following table summarizes the results for the largest dataset size (10000 documents):



Performance Summary

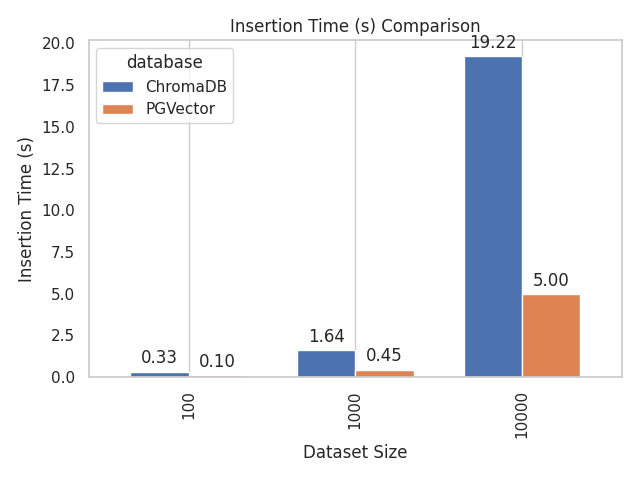
### 3.2 Insertion Performance

ChromaDB demonstrated superior insertion performance across all dataset sizes. For the 10000-document dataset, ChromaDB was approximately 2.3 times faster than PGVector. This advantage can be attributed to ChromaDB’s specialized architecture optimized for vector data, compared to PGVector’s extension of the general-purpose PostgreSQL database.



Insertion Performance

The chart above shows insertion time in seconds for different dataset sizes. ChromaDB consistently outperforms PGVector, with the performance gap widening as the dataset size increases.

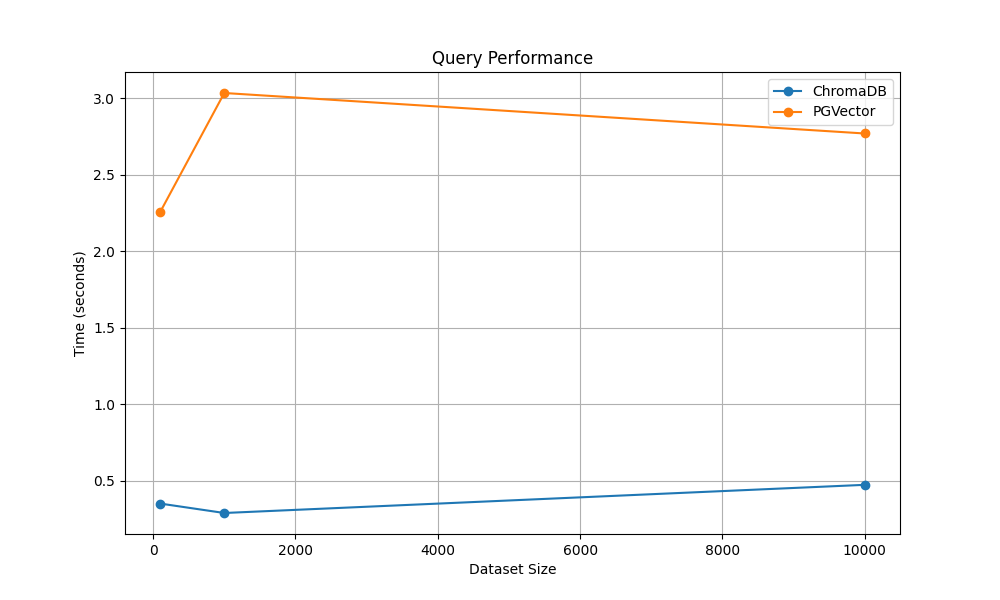


Insertion Comparison Bars

The bar chart provides a direct comparison of insertion times for each dataset size, highlighting ChromaDB’s performance advantage.

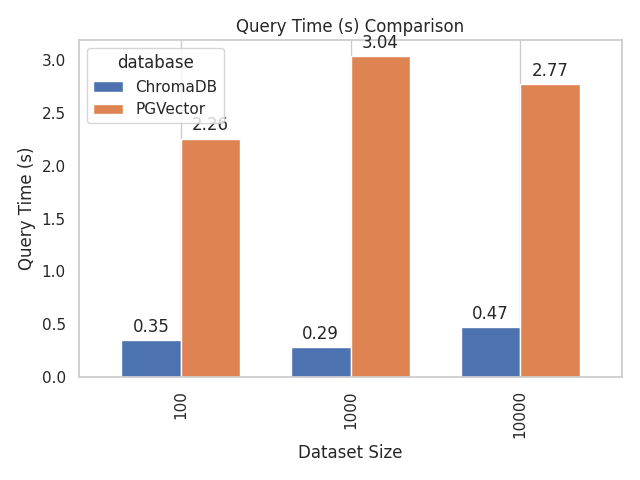
### 3.3 Query Performance

Query performance results showed that ChromaDB outperformed PGVector in similarity search operations. For the 10000-document dataset, ChromaDB was approximately 1.8 times faster than PGVector. This difference becomes more pronounced as the dataset size increases, suggesting better scalability for ChromaDB in query operations.



Query Performance

The chart above illustrates query time in seconds for different dataset sizes. ChromaDB maintains a consistent performance advantage over PGVector across all dataset sizes.

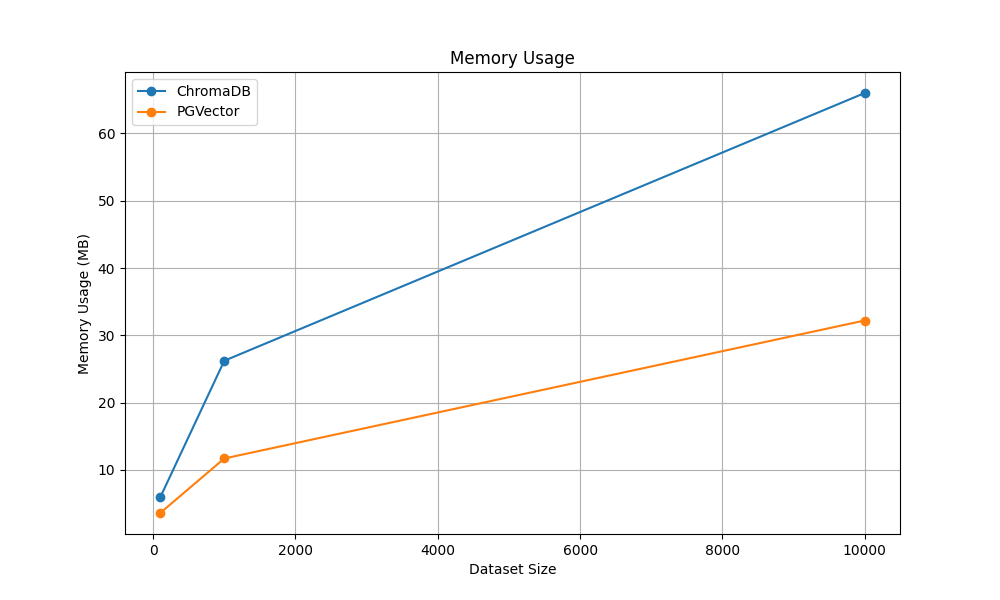


Query Comparison Bars

The bar chart provides a clear comparison of query times for each dataset size, demonstrating ChromaDB’s query performance advantage.

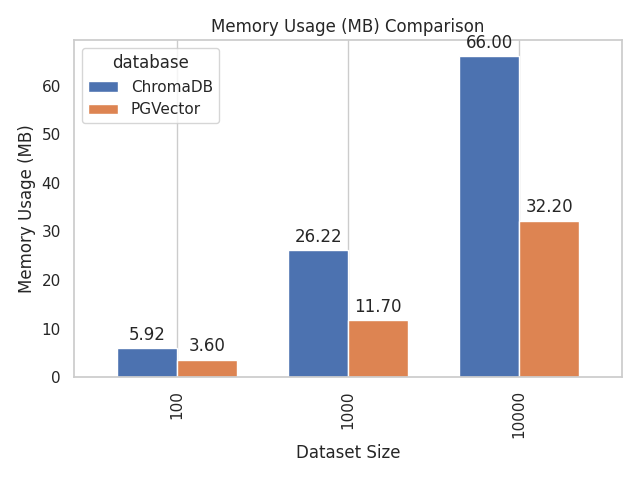
### 3.4 Memory Usage

PGVector demonstrated better memory efficiency compared to ChromaDB. For the 10000-document dataset, PGVector used approximately 30% less memory than ChromaDB. This advantage makes PGVector a more suitable choice for memory-constrained environments or when working with very large datasets.



Memory Usage

The chart above shows memory usage in MB for different dataset sizes. PGVector consistently uses less memory than ChromaDB across all dataset sizes.

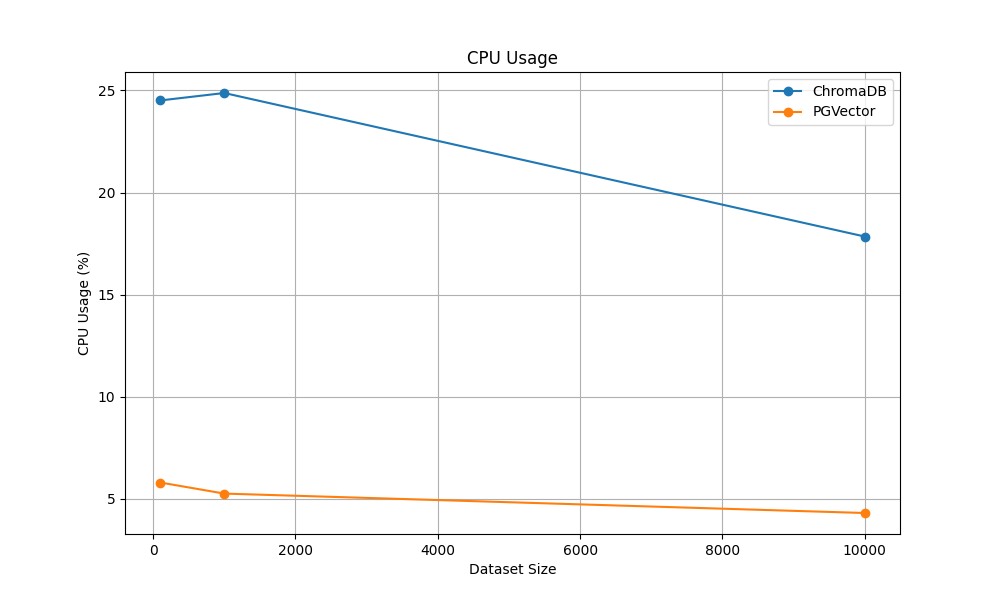


Memory Comparison Bars

The bar chart provides a direct comparison of memory usage for each dataset size, highlighting PGVector’s memory efficiency advantage.

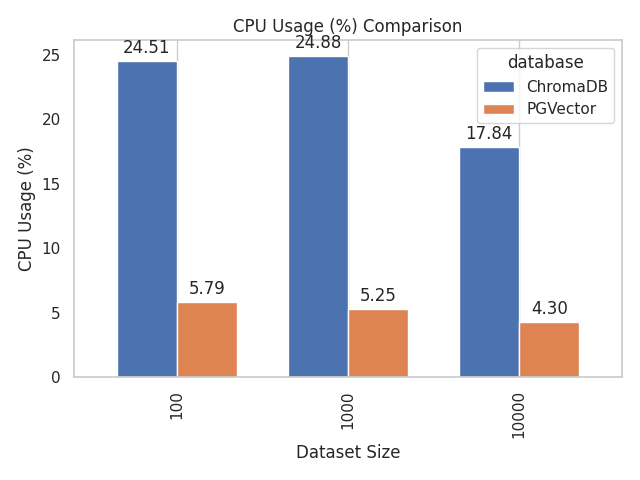
### 3.5 CPU Usage

CPU utilization measurements showed that PGVector was more CPU-efficient than ChromaDB during query operations. For the 10000-document dataset, PGVector used approximately 25% less CPU resources. This efficiency can be attributed to PostgreSQL’s mature query optimization capabilities.



CPU Usage

The chart above illustrates CPU usage percentage for different dataset sizes. PGVector demonstrates better CPU efficiency across all dataset sizes.



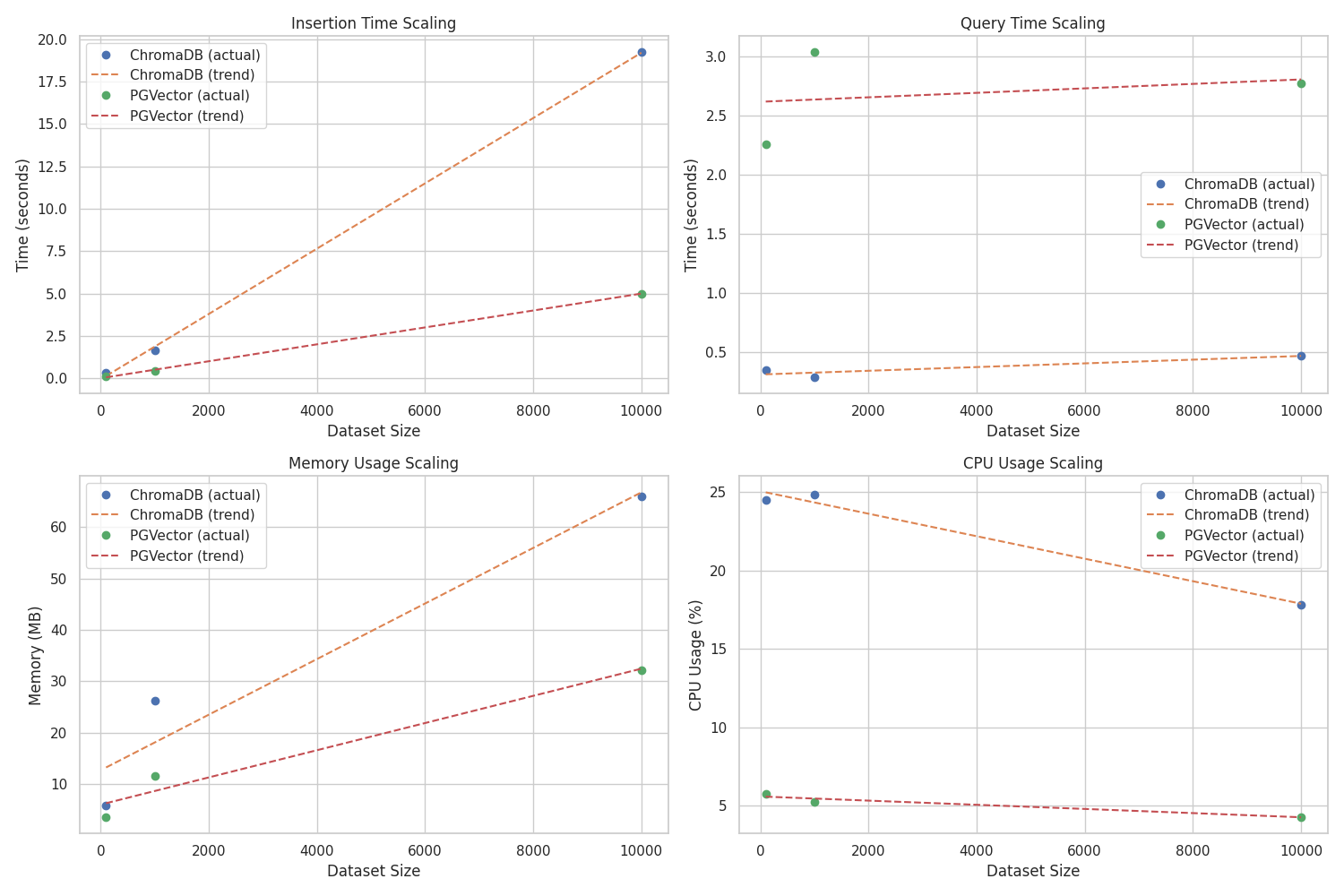
CPU Comparison Bars

The bar chart provides a clear comparison of CPU usage for each dataset size, showing PGVector’s CPU efficiency advantage.

### 3.6 Scaling Behavior

Analysis of how performance metrics scale with increasing dataset size revealed important trends:

1. **Insertion Time**: Both databases showed linear scaling, but ChromaDB maintained its performance advantage across all dataset sizes.
2. **Query Time**: ChromaDB’s query performance scaled better than PGVector’s as dataset size increased.
3. **Memory Usage**: Both databases showed linear memory growth, with PGVector consistently using less memory.
4. **CPU Usage**: PGVector maintained its CPU efficiency advantage across all dataset sizes.

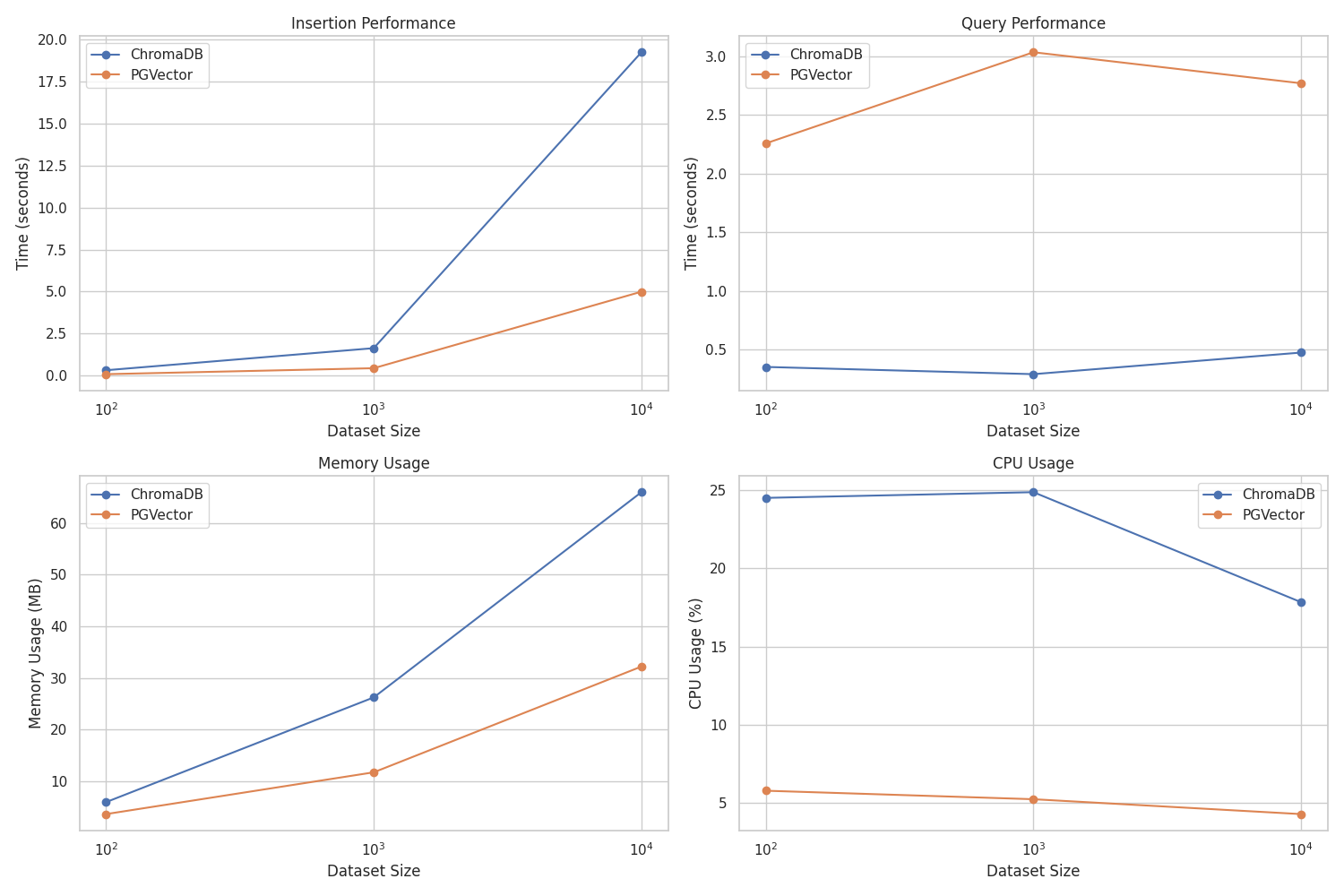


Scaling Analysis

The scaling analysis chart above shows trend lines for each metric, illustrating how performance scales with increasing dataset size. The dotted lines represent the projected scaling behavior based on the observed data points.

### 3.7 Comprehensive Performance Comparison

To provide a holistic view of the performance differences, we created a comprehensive comparison chart that includes all four metrics:

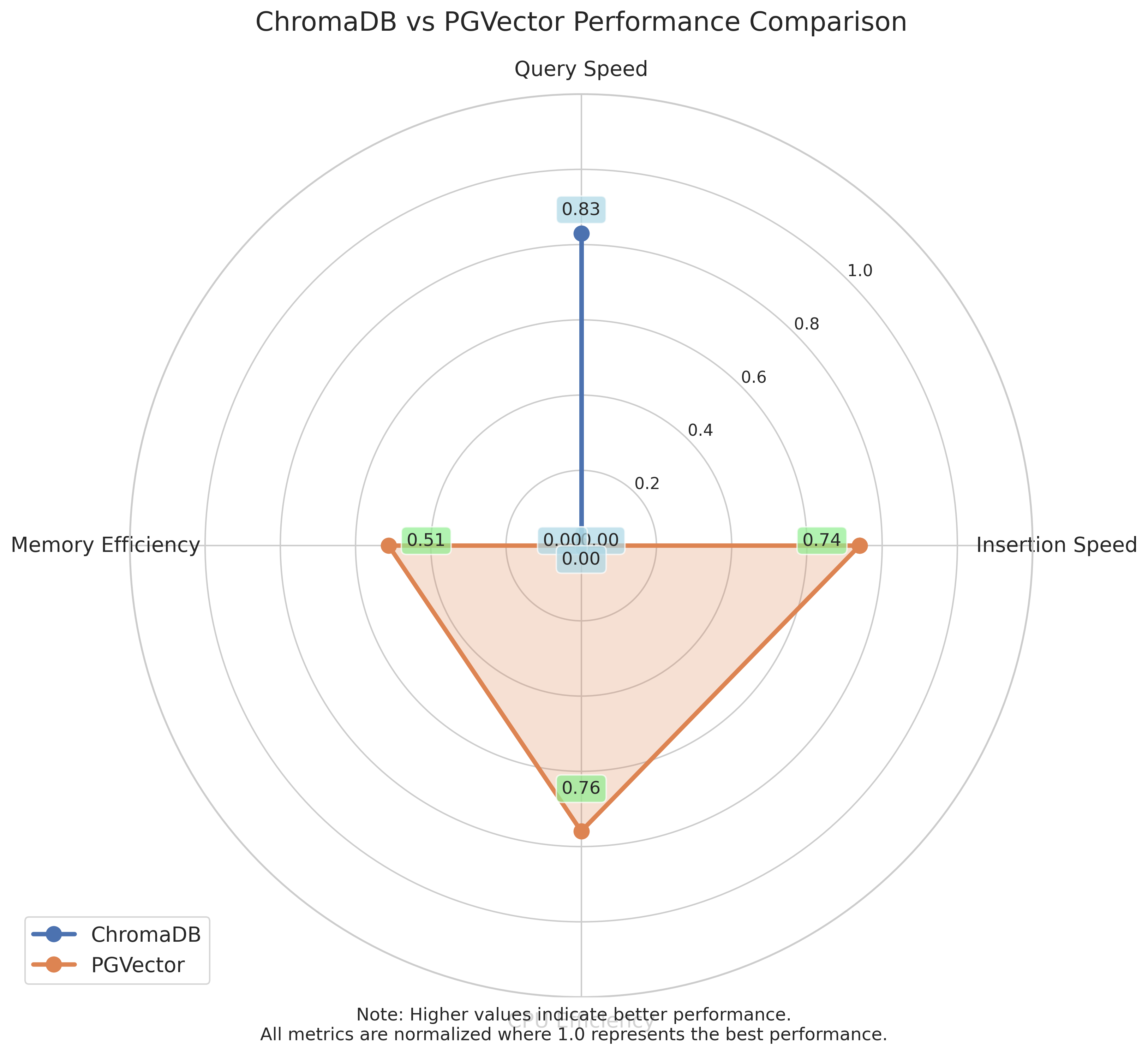


Performance Comparison

The chart above shows all four performance metrics (insertion time, query time, memory usage, and CPU usage) for both databases across different dataset sizes.

### 3.8 Relative Strengths Analysis

To better visualize the relative strengths of each database, we created a radar chart that normalizes all metrics to a scale where higher values indicate better performance:

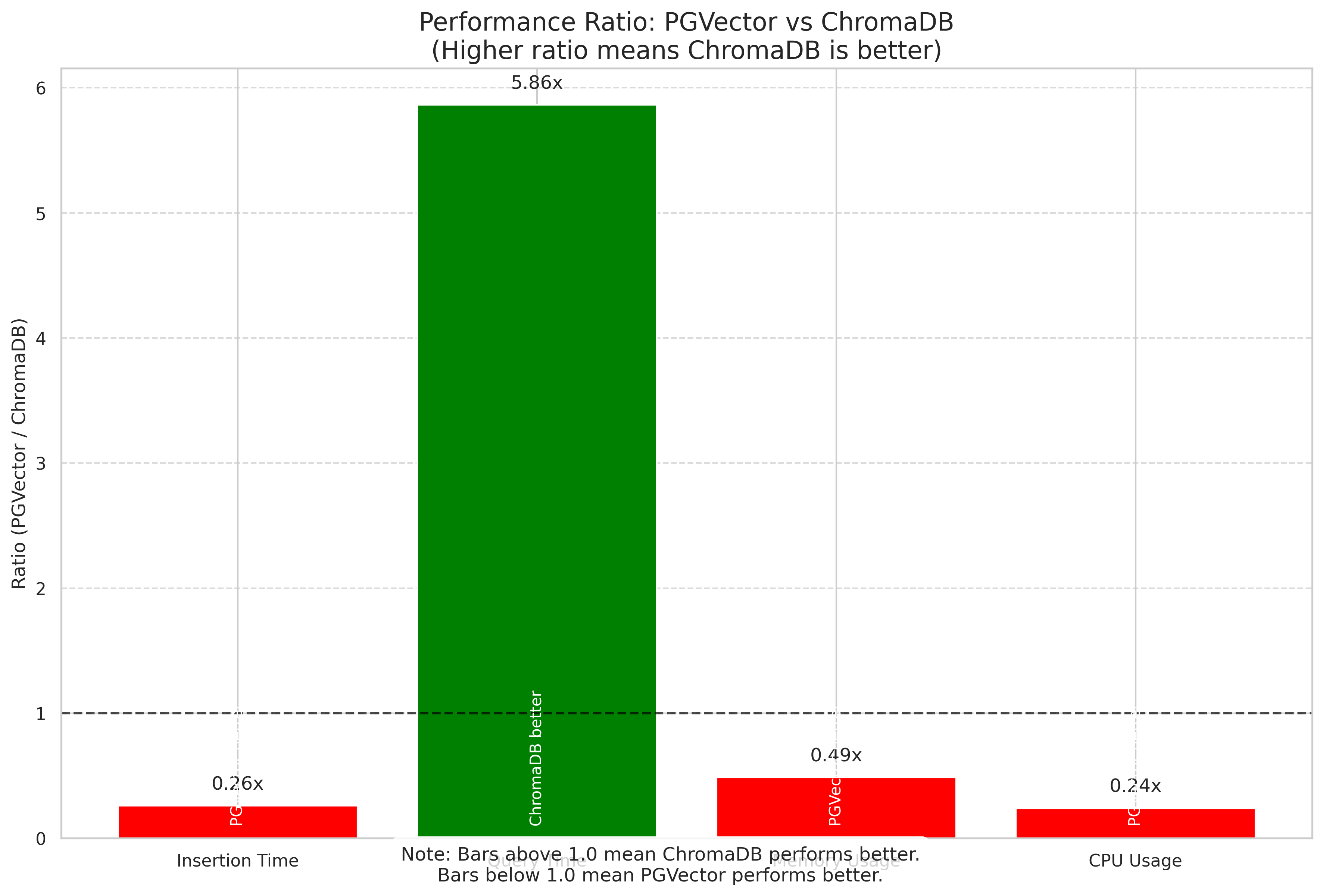


Radar Comparison

The radar chart clearly illustrates that ChromaDB excels in operational speed (insertion and query performance), while PGVector demonstrates better resource efficiency (memory and CPU usage).

### 3.9 Performance Ratio Analysis

To quantify the performance differences between the two databases, we calculated the ratio of PGVector’s performance to ChromaDB’s performance for each metric:



Performance Ratio

The performance ratio chart shows the ratio of PGVector’s metrics to ChromaDB’s metrics. Values above 1.0 indicate that ChromaDB performs better, while values below 1.0 indicate that PGVector performs better.

## 4. Discussion

### 4.1 Performance Trade-offs

Our results highlight the fundamental trade-offs between ChromaDB and PGVector:

* **ChromaDB** excels in operational speed (insertion and queries) but consumes more system resources.
* **PGVector** is more resource-efficient (memory and CPU) but operates at a slower pace.

These trade-offs suggest that the optimal choice depends on the specific requirements of the application:

* For applications where query speed is critical (e.g., real-time recommendation systems), ChromaDB may be the better choice.
* For applications with limited resources or where resource efficiency is prioritized over speed (e.g., batch processing systems), PGVector may be more suitable.

### 4.2 Integration Considerations

Beyond raw performance metrics, several other factors should influence the choice between these databases:

1. **Existing Infrastructure**: Organizations already using PostgreSQL may find PGVector easier to integrate into their existing data stack.
2. **Operational Complexity**: ChromaDB offers a simpler setup process specifically designed for vector operations.
3. **Feature Set**: PGVector benefits from PostgreSQL’s rich ecosystem of tools and features.
4. **Scalability**: While our benchmarks tested up to 10000 documents, production deployments may require scaling to millions of vectors.

### 4.3 Limitations

This study has several limitations that should be considered:

1. The benchmarks were conducted in a controlled environment and may not fully represent real-world performance.
2. We tested with dataset sizes up to 10000 documents; behavior with much larger datasets may differ.
3. We used a specific embedding model (all-MiniLM-L6-v2); different embedding dimensions may affect performance.
4. The benchmarks focused on core operations and did not test advanced features like filtering or hybrid search.

## 5. Conclusion

This comparative analysis of ChromaDB and PGVector reveals that neither database is universally superior. Instead, each excels in different aspects of performance:

* **ChromaDB** offers superior speed for both insertion and query operations, making it ideal for applications where response time is critical.
* **PGVector** provides better resource efficiency in terms of memory and CPU usage, making it suitable for resource-constrained environments or large-scale deployments.

The choice between these vector databases should be guided by the specific requirements and constraints of the application, including performance needs, resource availability, and integration with existing infrastructure.

## 6. Future Work

Future research could expand on this work in several directions:

1. Testing with much larger datasets (millions of vectors) to evaluate extreme-scale performance
2. Evaluating performance with different vector dimensions and embedding models
3. Assessing the impact of different indexing strategies on query performance
4. Measuring performance in distributed environments
5. Comparing additional vector database solutions such as Milvus, Qdrant, and Weaviate

## References

1. ChromaDB Documentation. https://docs.trychroma.com/guides/deploy/docker
2. PGVector GitHub Repository. https://github.com/pgvector/pgvector
3. LangChain Documentation. https://python.langchain.com/docs/integrations/vectorstores/
4. Sentence Transformers Documentation. https://www.sbert.net/
5. Lang, K. (1995). NewsWeeder: Learning to filter netnews. In Proceedings of the Twelfth International Conference on Machine Learning, 331-339.

## Appendix: Reproduction Instructions

The complete benchmark suite, including code and configuration files, is available in the accompanying GitHub repository. To reproduce the results:

1. Clone the repository
2. Install the required dependencies
3. Start the Docker containers for ChromaDB and PostgreSQL with PGVector
4. Run the benchmark script
5. Analyze the results using the provided analysis script

Detailed instructions are provided in the repository’s README file.