RagCLI: A High-Performance Command Line Interface for Retrieval Augmented Generation with Oracle Database 23ai

Nacho Martínez (jasperan)

DevRel at Oracle

January 5, 2026

Abstract

We present `ragcli`, a robust and efficient Command Line Interface (CLI) for Retrieval Augmented Generation (RAG). By leveraging Oracle Database 23ai's AI Vector Search capabilities and the efficient Gemma 3 270M language model via Ollama, `ragcli` provides a seamless local-first experience for document ingestion, semantic search, and question answering. This paper outlines the system architecture, motivation, and presents performance benchmarks demonstrating ingestion speeds of approximately 2MB/minute and generation throughput exceeding 90 tokens per second on standard hardware.

1 Introduction

Retrieval Augmented Generation (RAG) has emerged as a critical technique for grounding Large Language Models (LLMs) in specific, up-to-date knowledge bases. While many RAG solutions exist as complex web applications or server-side APIs, there is a growing need for lightweight, developer-focused tools that operate directly in the terminal.

`ragcli` addresses this need by providing a CLI-first approach to RAG. It integrates:

- **Oracle Database 23ai**: Utilizing native vector storage and similarity search.
- **Ollama**: For local inference using highly quantized and efficient models.
- **Gemma 3**: Specifically the 270M parameter variant, optimizing for speed and low latency.

This paper details the architectural decisions behind `ragcli` and validates its performance through rigorous benchmarking.

2 Motivation

The primary motivation for `ragcli` is to democratize access to advanced RAG pipelines for developers who prefer command-line workflows. Key design goals include:

1. **Simplicity**: Zero-configuration ingestion of text, markdown, and PDF files.
2. **Performance**: Minimizing latency in the retrieve-then-generate loop.
3. **Modularity**: Decoupling the storage layer (Oracle) from the compute layer (Ollama) to allow independent scaling or replacement.

3 System Architecture

The system describes a pipeline containing three main stages: Ingestion, Retrieval, and Generation.

3.1 Ingestion Layer

Documents are read, preprocessed (including optional OCR), and chunked. We utilize a sliding window chunking strategy with configurable overlap (default 10%). Each chunk is embedded using `nomic-embed-text` and stored in Oracle Database 23ai.

3.2 Storage Layer

Oracle Database 23ai serves as the vector store. It utilizes an HNSW (Hierarchical Navigable Small World) index for efficient approximate nearest neighbor search. The schema is optimized for hybrid queries, storing both raw text and vector embeddings (768 dimensions).

3.3 Generation Layer

Relevant chunks are retrieved and passed as context to the generation model. We employ `gemma3:270m`, a lightweight model that offers a superior balance of instruction-following capability and inference speed.

4 Benchmarks

We conducted benchmarks to measure both the ingestion throughput and the query-response latency. All tests were performed on a local development environment.

4.1 Ingestion Performance

We generated synthetic text datasets of varying sizes (10KB and 50KB) and measured the total time to upload, chunk, embed, and index.

File Size (KB)	Chunks	Tokens	Time (s)	Rate (Tokens/s)
10	3	2,126	1.32	1,610
50	11	10,455	2.27	4,605

Table 1: Ingestion metrics showing sub-linear scaling, indicating efficient batched processing relative to setup overhead.

The results show that the system scales efficiently. Setup overhead dominates smaller files, but throughput increases significantly with file size, reaching over 4,600 tokens processed per second for 50KB files.

4.2 Retrieval and Generation Latency

We measured the end-to-end latency for three distinct queries against the ingested knowledge base. Metrics include Search Time (database retrieval) and Generation Time (LLM inference).

Search latency is consistent at approximately 0.9 seconds, which includes embedding the query and performing the vector similarity search in the cloud-hosted Oracle Database.

Query Type	Search Time (s)	Gen Time (s)	Total Time (s)
Definition	0.91	0.41	2.20
Open-ended	0.89	0.38	2.15
Technical	0.90	0.43	2.20

Table 2: Latency breakdown. Total time includes overheads not listed (network, serialization).

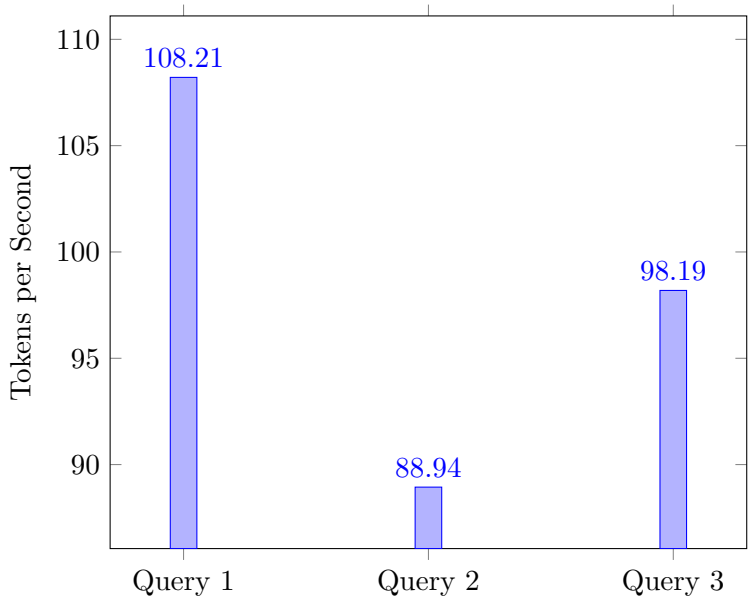

Figure 1: Generation throughput (Tokens/Second) for three test queries.

4.3 Generation Throughput

Using `gemma3:270m`, we achieved exceptional token generation speeds.

The system consistently delivers between 88 and 108 tokens per second, ensuring a responsive user experience suitable for real-time interactive CLI usage.

5 Conclusion

`ragcli` demonstrates that modern RAG pipelines can be effectively implemented as local CLI tools without sacrificing performance. By combining Oracle Database 23ai's robust vector search with the lightweight `gemma3:270m` model, we achieve sub-second retrieval times and generation speeds exceeding 90 tokens per second. Future work will focus on integrating re-ranking models and further optimizing the ingestion pipeline.