Bachelor Project Presentation

Efficient Text Embedding Inference in Data Streaming

Raphael Lachtner, WS 2024/25

Introduction

- Growing need for text understanding in data streaming systems
- Machine learning inference can be challenging
- Especially in distributed, (near-) real time solutions like Apache Kafka

Research Gap

- Lack of standardized methods for ML inference in Kafka
- No efficient way to generate text embeddings within Kafka pipelines
- Need for a Kafka-native solution
  - Must be flexible, resource-efficient, scalable
  - Seamless integration with Kafka
  - Support for error handling, delivery semantics, autoscaling, data seriailzation

Text Embeddings

- Converts text into numerical vectors
- Captures meaning and relationships
- Similar concepts are close in vector space
- Helpful for natural language understanding

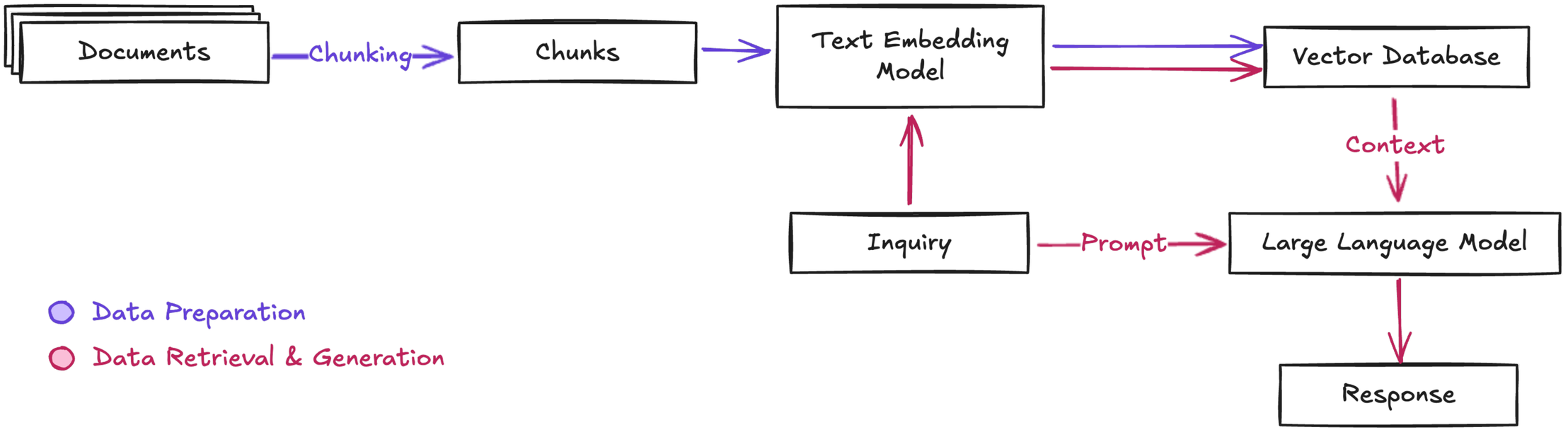
Applications

- Semantic searches
- Information retrieval
- Recommendation systems
- Text classification

Retrieval-Augmented Generation (RAG)

→ Enhances large language models by including query specific knowledge into generation process

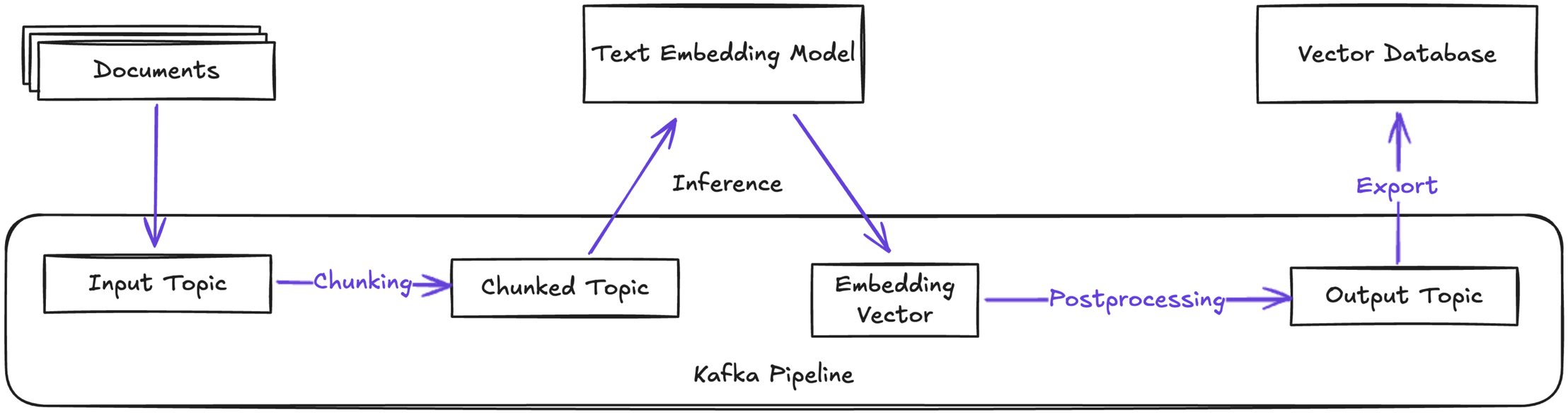
→ Retrieves relevant information from a knowledge base



RAG Pipeline in Apache Kafka

Extent of this project:

Framework for data preparation step that embeds data in a Kafka pipeline



Solution Overview

Simple to use but powerful and flexible text embedding framework, that handles:

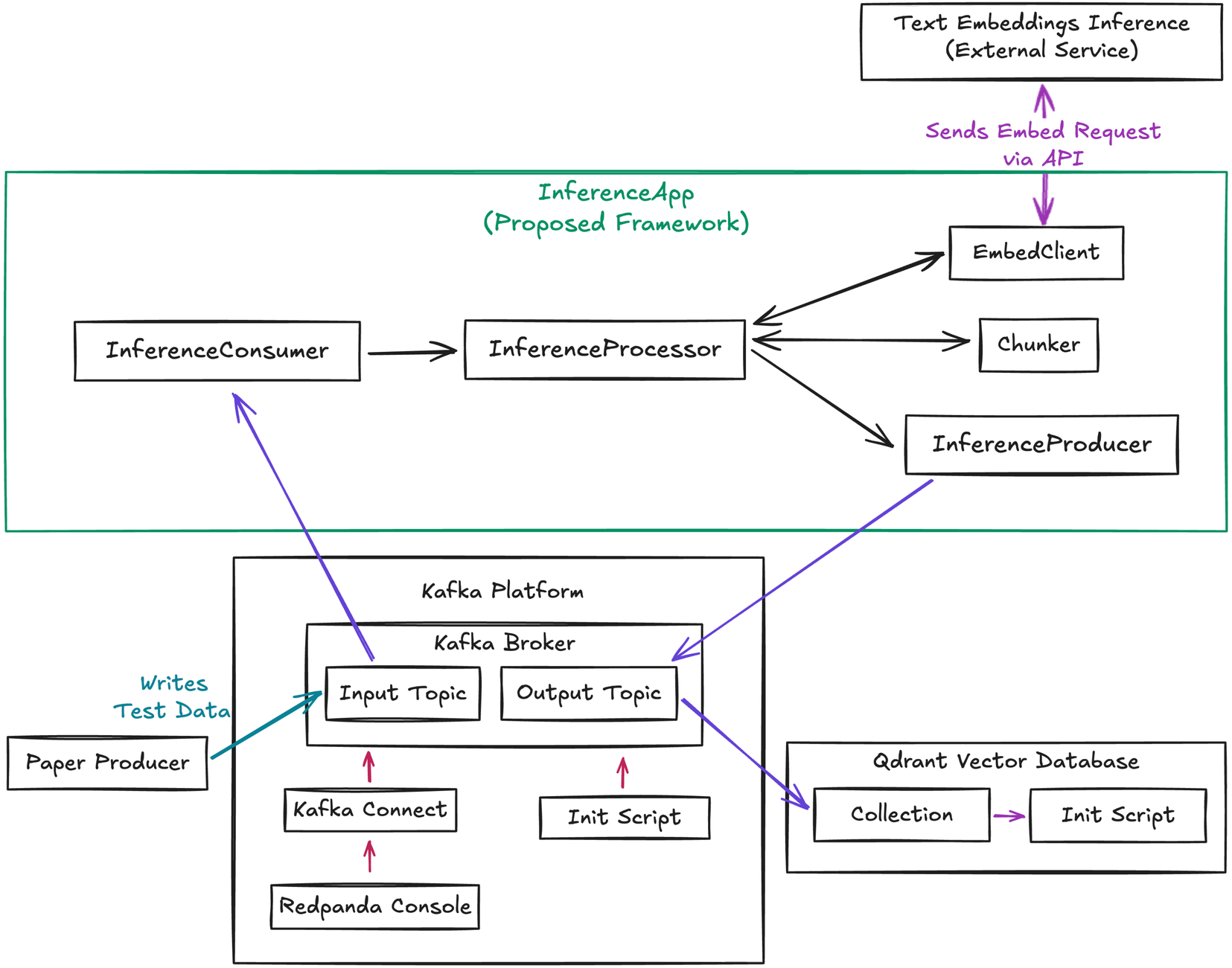
- Message consumption
- Text chunking
- Efficient batching
- Efficient GPU-accelerated inference
- Message production

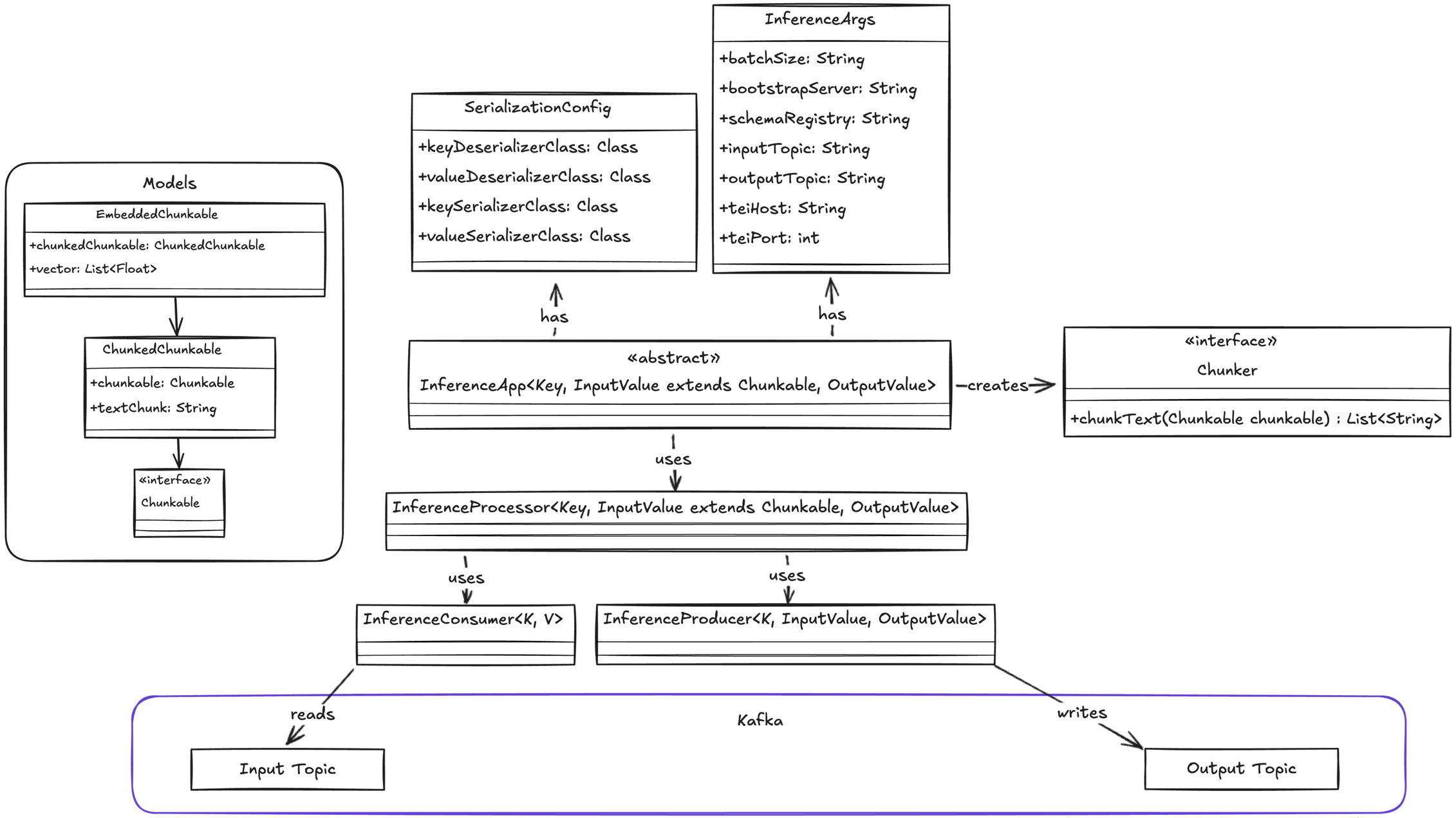
Features

- Configurable pipeline
- Support for different models
- Customizable chunking strategy
- Simple to build and extend pipelines
- Focus on performance and resource efficiency

Technical Deep Dive

- Uses highly optimized Hugging Face text-embeddings inference service
- Calls gRPC API to efficiently embed text
- Utilizes official Kafka Java APIs: Consumer API and Producer API
- Chunking strategy implementable (e.g. langchain4j)
- Read and chunk message in batches
- Asynchronously send them to inference service
- Returned data is post-processed and written back to Kafka


Text Embeddings Inference
(External Service)
Sends Embed Request
via API
InferenceApp
(Proposed Framework)
EmbedClient
InferenceConsumer
InferenceProcessor
Chunker
InferenceProducer
Kafka Platform
Kafka Broker
Writes
Test Data
Input Topic
Output Topic
Paper Producer
Kafka Connect
Init Script
Redpanda Console
Qdrant Vector Database
Collection
Init Script


InferenceArgs
+batchSize: String
+bootstrapServer: String
+schemaRegistry: String
+inputTopic: String
+outputTopic: String
+teiHost: String
+teiPort: int
SerializationConfig
+keyDeserializerClass: Class
+valueDeserializerClass: Class
+keySerializerClass: Class
+valueSerializerClass: Class
Models
EmbeddedChunkable
+chunkedChunkable: ChunkedChunkable
+vector: List<Float>
ChunkedChunkable
+chunkable: Chunkable
+textChunk: String
«interface»
Chunkable
has
has
«abstract»
InferenceApp<Key, InputValue extends Chunkable, OutputValue>
creates
«interface»
Chunker
+chunkText(Chunkable chunkable) : List<String>
uses
InferenceProcessor<Key, InputValue extends Chunkable, OutputValue>
uses
uses
InferenceConsumer<K, V>
InferenceProducer<K, InputValue, OutputValue>
reads
writes
Kafka
Input Topic
Output Topic

Comparing Implemented Approaches

Kafka Client options:

- Kafka Streams: High-level abstraction, simple to implement powerful applications
- Python Client: Established machine learning ecosystem, performance trade-offs
- Consumer/Producer API: Low-level control

Model Serving options:

- External
- Co-located

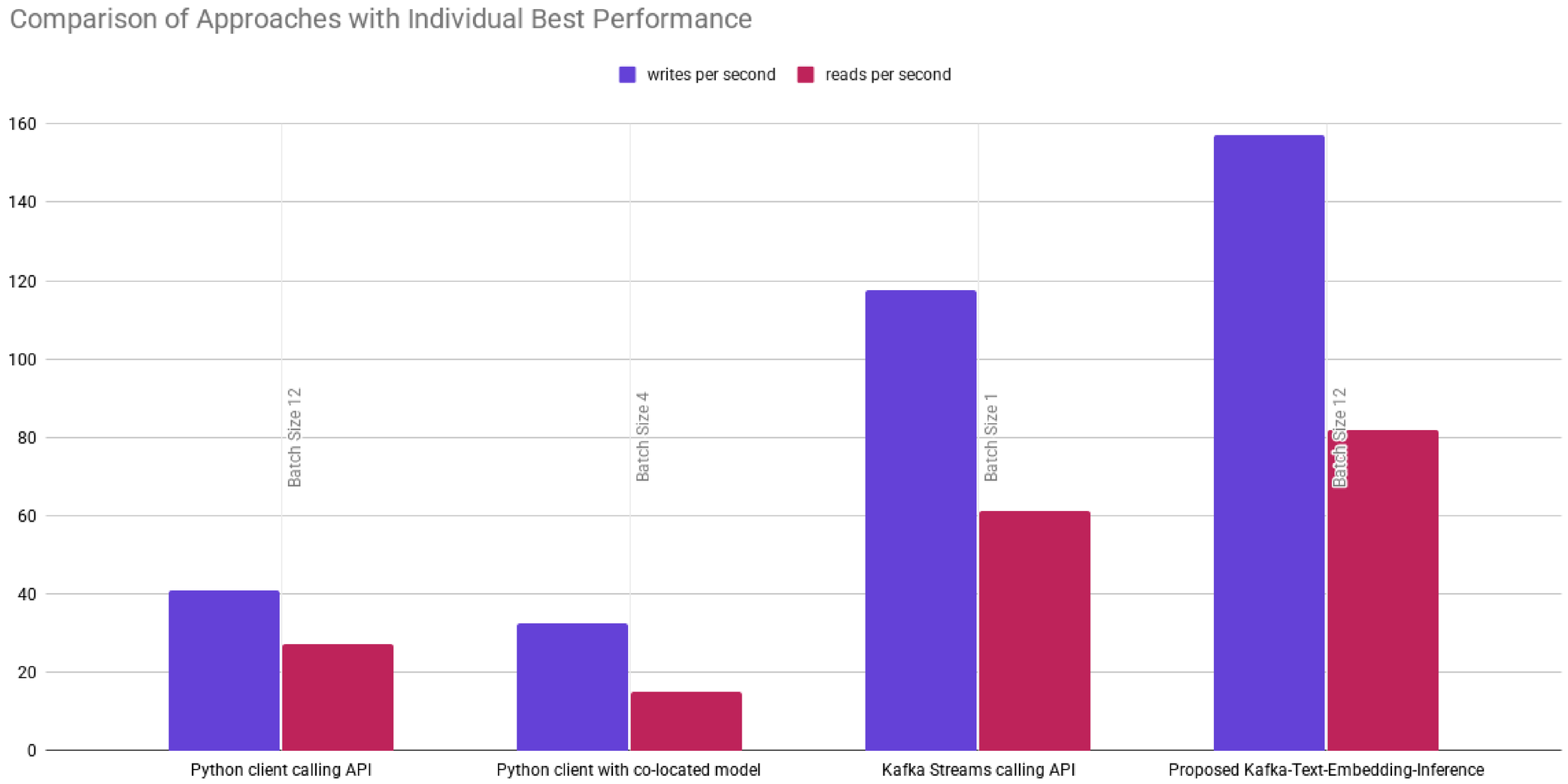
Tested Approaches

Python Kafka client calling external inference service

Python Kafka client with co-located model

Kafka Streams client with external inference service

Proposed framework, calling external inference service



High Throughput (writes per second) is crucial for processing large volumes of data efficiently.

Lessons Learned & Evaluation

- Batch Size Optimization
- Client Implementation
- External vs. Embedded Models
- GPU Resource Management

Evaluation Summary - Proposed Framework Advantages:

- ~33% higher throughput with optimized batching
- Highest throughput of all tested approaches
- Reduced container size and startup time
- Simplified embedding pipeline building

References

Horchidan A, Chen Y, Boncz P and Raasveldt M (2024). Crayfish: Navigating the Labyrinth of Machine Learning Inference in Stream Processing Systems. In: Proceedings of the 27th International Conference on Extending Database Technology (EDBT). EDBT/ICDT 2024 Joint Conference, March 25-28, 2024, Lisbon, Portugal

"LangChain4j RAG (Retrieval-Augmented Generation)" Accessed: Jan. 22, 2025. [Online]. Available https://docs.langchain4j.dev/tutorials/rag/

"Apache Kafka" Accessed: Jan. 22, 2025. [Online]. Available: https://kafka.apache.org/powered-by

"Huggingface/Text-Embeddings-Inference" Accessed: Jan. 22, 2025. [Online]. Available: https://github.com/huggingface/text-embeddings-inference

C. Martín, P. Langendoerfer, P. S. Zarrin, M. Díaz, and B. Rubio, "Kafka-ML: Connecting the Data Stream with ML/AI Frameworks," Future Generation Computer Systems, vol. 126, pp. 15–33, Jan. 2022, doi: 10.1016/j.future.2021.07.037.

A. Farki and E. A. Noughabi, "Real-Time Blood Pressure Prediction Using Apache Spark and Kafka Machine Learning," in 2023 9th International Conference on Web Research (ICWR), Tehran, Iran, Islamic Republic of: IEEE, May 2023, pp. 161–166. doi: 10.1109/ICWR57742.2023.10138962.

"Qdrant/Fastembed." Accessed: Jan. 22, 2025. [Online]. Available: https://github.com/qdrant/fastembed

"How Kafka Streams Work and Their Key Benefits." Accessed: Jan. 22, 2025. [Online]. Available: https://double.cloud/blog/posts/2024/05/kafka-streams/