Domain Identification in Work Reports Using Transformers

A Comprehensive NLP Solution with Hybrid Architecture and Production Deployment

Manas Tomar

Abstract

01	The choice of the models used, and the correct hybrid pipeline
02	Deployment on Nvidia Triton Inference Server
03	Providing an interface like FastApi

Overview

PROJECT

Problem Statement

RESEARCH CHALLENGE

Core Problem

- **Automatic domain identification in technical work reports**
- **There is currently no direct domain identification model available on Hugging Face**

Does something feel obvious?
We use generative LLMs in our everyday lives, out of which many
are now open sourced.

As open source models advance so much this looks like the obvious solution,
I will address this shortly

DATA SOURCING STRATEGY - PRIMARY SOURCE SELECTION

WE NEED TO EXTARCT DOMAINS FROM WORK REPORTS, BUT BEFORE EXTRACTING THESE DOMAINS WE SHOULD FIRST CLASSIFY ATLEAST THE POSSIBLE DOMAINS WHICH CAN SERVE AS OUR LABELS

EBSCO APPLIED SCIENCE & TECHNOLOGY SOURCE ULTIMATE

Criterion	Specification	Benefit
Coverage	2000+ publications	Comprehensive domain representation
Quality	Peer-reviewed content	Reliable domain labels
Structure	Taxonomical consistency	Automated processing compatibility
Specialization	Applied sciences focus	Work report context alignment

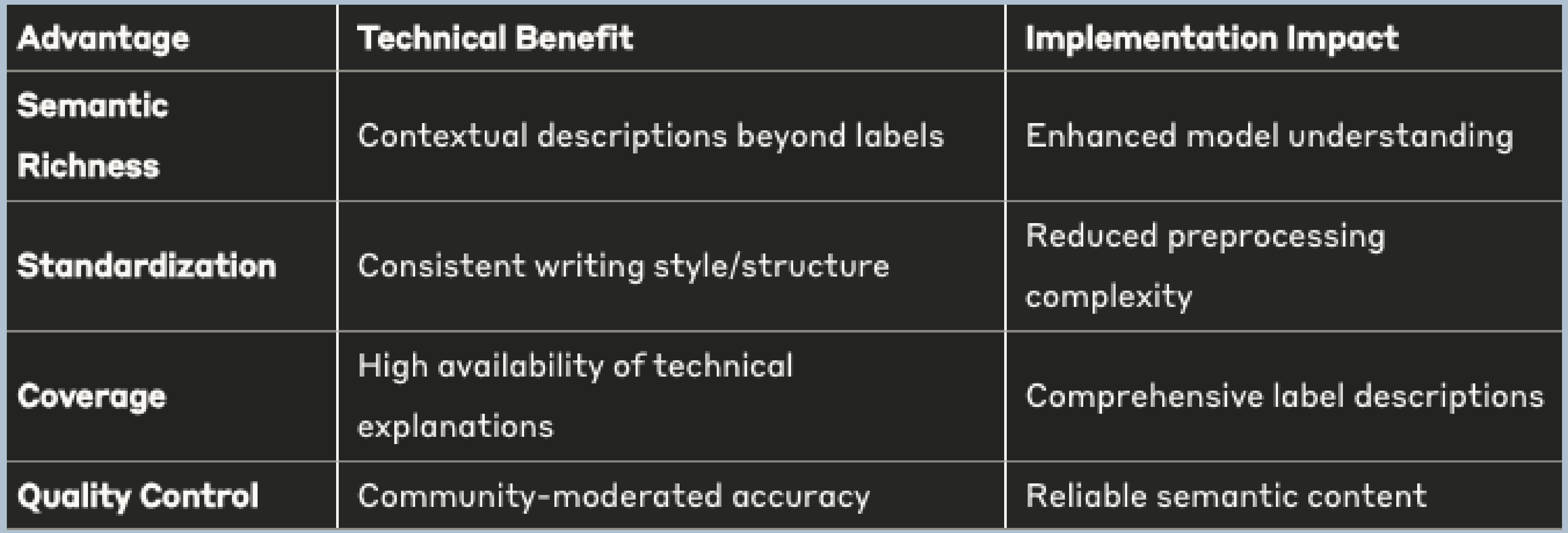
Data Quality Metrics

- Publication Types: Journals, conference proceedings, technical reports
- Domain Breadth: Engineering, technology, applied sciences
- Temporal Coverage: Current and historical content
- Language: Primarily English with standardized terminology

DATA ENHANCEMENT - WIKIPEDIA API INTEGRATION

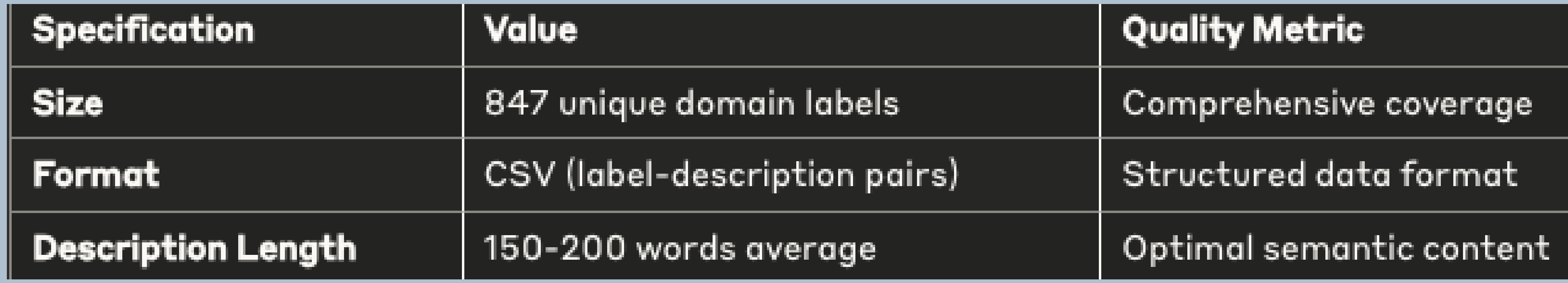
LABEL DESCRIPTION GENERATION METHODOLOGY
WIKIPEDIA API INTEGRATION STRATEGY

JUSTIFICATION FOR WIKIPEDIA INTEGRATION



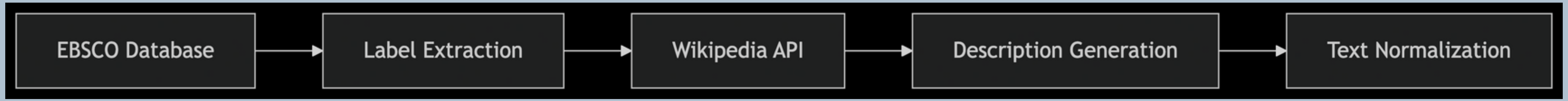
Advantage	Technical Benefit	Implementation Impact
Semantic Richness	Contextual descriptions beyond labels	Enhanced model understanding
Standardization	Consistent writing style/structure	Reduced preprocessing complexity
Coverage	High availability of technical explanations	Comprehensive label descriptions
Quality Control	Community-moderated accuracy	Reliable semantic content

FINAL DATA SET CHARACTERISTICS



Specification	Value	Quality Metric
Size	847 unique domain labels	Comprehensive coverage
Format	CSV (label-description pairs)	Structured data format
Description Length	150-200 words average	Optimal semantic content

COMPREHENSIVE DATA TRANSFORMATION WORKFLOW



*text normalization is a step but not carried out yet
as the data was pretty clean and semantically rich

METHODOLOGY

THERE ARE MULTIPLE WAYS TO GET THE DOMAIN,LETS GET INTO IT

1 Zero-Shot CLassification

- Implementation: Pre-trained models, no fine-tuning
- Advantages: No training required, quick deployment
- Limitations: Limited accuracy on specialized domains

2 Cosine Similarity with Bi-Encoders

- Implementation: Dense vector representations
- Advantages: Efficient computation, scalable
- Limitations: Limited query-candidate interaction

3 Cross-Encoders

- Implementation: Joint encoding of query-candidate pairs
- Advantages: Superior accuracy, nuanced understanding
- Limitations: Computational overhead, slower inference

HYBRID ARCHITECTURE DESIGN

01

Stage 1: Efficient Retrieval

- Pre-computed label embeddings in PyTorch tensors
- Vectorized similarity computations
- Caching for frequent queries

02

Stage 2: Precision Reranking

- Batch processing for cross-encoder inference
- Optimized attention mechanisms

The system architecture processes work report text by converting it into dense vector embeddings, enabling fast retrieval through cosine similarity search. This initial stage identifies the top 10 candidate matches relevant to the input. To enhance precision, a cross-encoder then re-evaluates these candidates by jointly analyzing each query-candidate pair, capturing deeper semantic relationships. This two-step approach ensures a balance between retrieval efficiency and high-quality, accurate final rankings for domain identification.

MODEL SELECTION - BI-ENCODER

The project settles on all-mpnet-base-v2, its a well documented model with decent performance on reports, some other tried models and their drawbacks given below... is the current model future proof?

1
MPNET-V2

This model offers excellent quality with a good balance of speed and memory usage, delivering fast inference at around 12.3ms and requiring about 420MB. It consistently performs well on semantic similarity tasks, making it suitable for technical domains.

2
MINI-LM-V6

A lightweight option with very low memory usage of only 90MB and fast inference at 8.7ms. It provides good quality but slightly lower accuracy than larger models, making it suitable for resource-constrained environments.

3
SPECTER-2

Tailored for scientific and technical text, SciBERT provides excellent quality with an inference time of 18.5ms and a memory usage of about 440MB. It’s ideal for specialized domains but is slightly slower than lighter models.

4
NOMICAI

Delivers very good quality embeddings with a moderate inference time of 15.2ms and a larger memory footprint of around 550MB. It’s a strong choice for applications requiring robust semantic representation.

The inference speeds are average case speed and not measured according to this project

MODEL SELECTION CROSS-ENCODER

I DECIDED ON BGE-RERANKER-LARGE, A STATE OF THE ART MODEL WITH HIGH PERFOMANCE SCORE, ALTHOUGH THIS DOES NOT GURANTEE HIGH PERFOMANCE ON BARC WORK REPORTS

1

bge-reranker-large

A high-accuracy cross-encoder designed for reranking, BGE-Reranker-Large uses an embedding size of 1024, enabling it to capture fine-grained semantic interactions in technical text. It handles longer input sequences (up to 512 tokens) and is well-suited for domain-rich tasks. While the inference time averages around 60–80ms per pair, the quality is excellent, with a memory footprint of ~1.4GB. This model excels in tasks where precision is more critical than latency.

2

MiniLM

A compact and efficient model with an embedding size of 384, MiniLM offers much faster inference (~8–10ms per pair) and low memory usage (~110MB). While not as semantically deep as BGE-Reranker, it provides reliable results for many reranking tasks with limited computational overhead. It's ideal for high-throughput scenarios or where latency is a key constraint.

Using Generative LLMs like Gemma for Domain Identification

Now we get back to the question, cant we just use open sourced extremley powerful LLMs?

01

- Adapts to unseen domains without retraining
- Generates explanatory output alongside predictions

02

- Models act as black boxes, with limited interpretability
- High risk of hallucinations, where the model generates plausible but incorrect information
- Lacks calibrated confidence scores, complicating evaluation
- Sensitive to prompt design, leading to variability in results

03

So what do we do? This project aims to construct a hybrid pipeline using Cross-encoders and Bi-encoders.

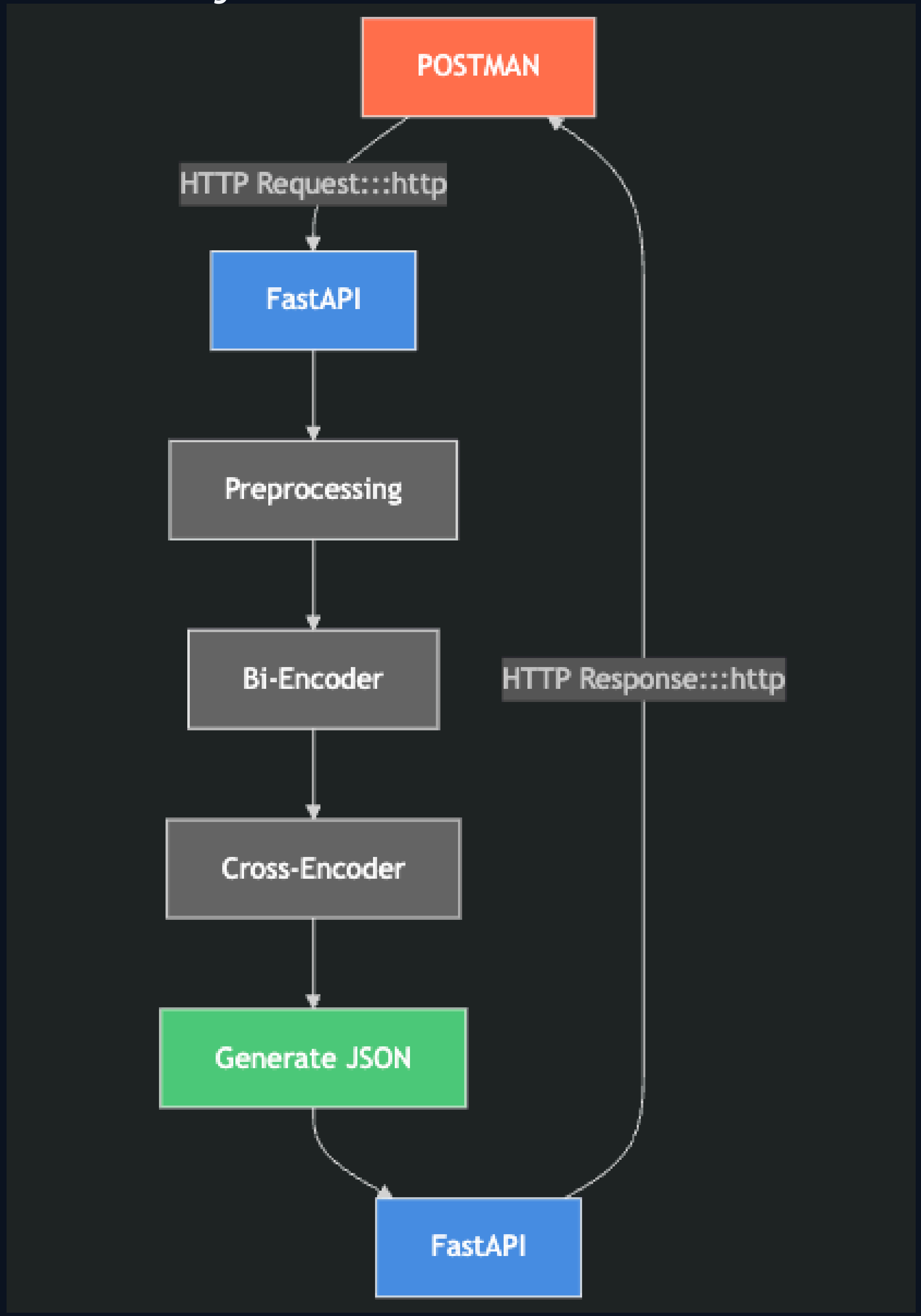
API DEPLOYMENT WITH FASTAPI & UVICORN

The entire domain identification pipeline is deployed as a unified web service using FastAPI, a modern Python framework that enables the creation of high-performance APIs, served via Uvicorn, an ASGI (Asynchronous Server Gateway Interface) server optimized for concurrency and low-latency responses. This setup allows the system to handle POST requests from tools like Postman, supporting functionalities such as customizable top_k results, JSON batch inputs for processing multiple reports simultaneously, and secure access through password-protected API keys, ensuring a scalable, flexible, and production-ready deployment environment.

Your paragraph text

Is this deployment technique sufficient ? What happens if we get multiple request?

any HTTP client like Postman



Both of the fastAPI boxes represent the same webserver

WHY DO WE NEED TRITON?

MEMORY EFFICIENCY

MULTI-MODAL PIPELINES

DYNAMIC BATCHING

IN SHORT-

Triton handles optimized model execution and scheduling, while FastAPI is limited to serving Python logic. Triton is essential when deploying multi-model, high-throughput systems requiring parallelism, batching, and real-time performance at production scale.

DEPLOYMENT STRATEGY

SCALABLE DEPLOYMENT WITH NVIDIA TRITON INFERENCE SERVER

To support efficient, scalable, and production-grade domain identification, the pipeline is deployed using NVIDIA Triton Inference Server instead of relying solely on FastAPI with Uvicorn. While FastAPI is ideal for lightweight API serving, Triton offers advanced features essential for ML model orchestration at scale—such as dynamic batching, concurrent model execution, and native multi-GPU support.

01

Performance Optimization
- Dynamic Batching: Combines small requests into efficient batches
- Model Versioning: Supports A/B testing and easy rollback

02

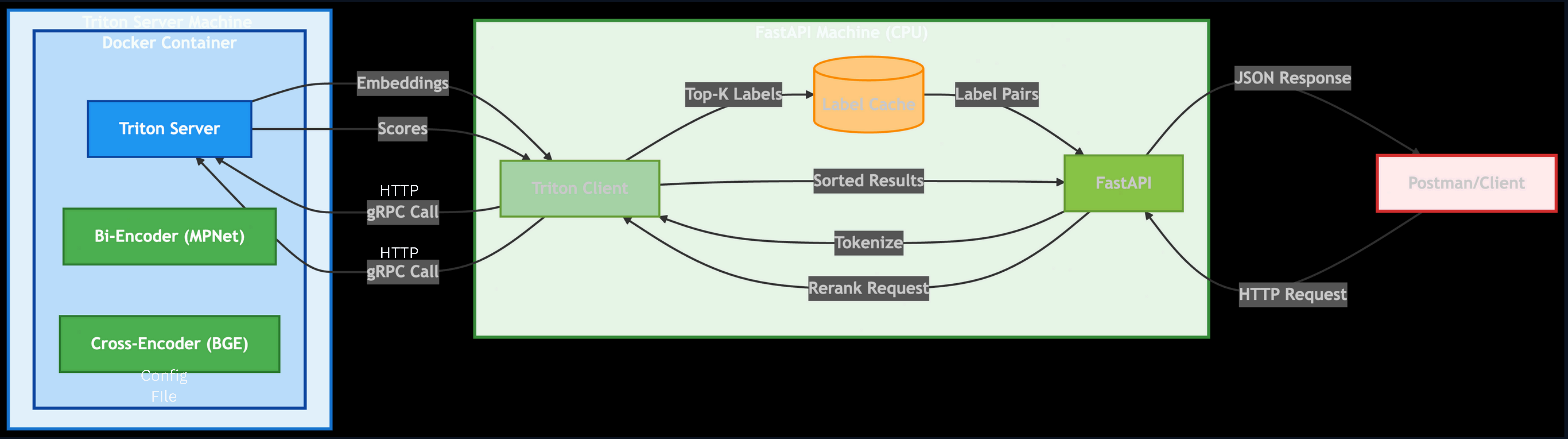
Production-Ready Capabilities
- Health Monitoring: Built-in metrics, health checks
- Load Balancing: Smart request routing

03

Framework & Pipeline Compatibility
- Native PyTorch & ONNX support
- Python backends for preprocessing logic

The configuration and flow diagram

The docker container is made by running the docker image of the triton server and then the model.pt (torchscript) files are attached to it



Going forward

01

The models used right now maynot be the best models for tommorow.

02

The Triton inference server allows us to easily replace models and hence we can make this project relevant for future possibilities.

03

That will be it for this Presentation.

Thank you very much!

- FastAPI alone: Multiple worker processes → each loads its own model copy → high RAM/VRAM usage.
- Triton + FastAPI: Loads model once → shared across all requests → saves memory

- FastAPI alone: Manually manage calls between different models.
- Triton: Supports ensemble models → easily chains multiple models (e.g., bi-encoder + cross-encoder) in one pipeline.

- FastAPI alone: Processes requests one-by-one → inefficient GPU utilization.
- Triton: Combines small requests into batches → better throughput and cost efficiency.