# Transformers for our problems

[Nareli] — AI Applications Specialist

September 26, 2025

# Core Idea of Transformers

- **Architecture:** Based on self-attention, allowing models to weigh relationships across all input tokens in parallel.
- **Advantage over RNNs/CNNs:** Handles long-range dependencies and scales better.
- **Applications:**
  - Text: summarization, compliance, search
  - Code: bug detection, code generation
  - Time series: forecasting, anomaly detection
  - Multimodal: integrating text, images, logs

# Advantages for Enterprise Efficiency

- **Automation of Knowledge Work:** Report drafting, ticket classification.
- **Decision Support:** Context-aware recommendations, anomaly detection.
- **Scalability:** Pretrained models reduce data requirements.
- **Integration:** Can be paired with databases, knowledge graphs, vector search (RAG).
- **Adaptability:** One architecture across multiple modalities.

# Difficulties: Knowledge

- Need deeper understanding of transformer mechanisms (attention, embeddings).
- Evaluation metrics: perplexity, BLEU, ROUGE, accuracy.
- Awareness of limitations: hallucinations, bias, lack of transparency.

# Difficulties: Skills

- **MLOps:** Deployment, monitoring drift, latency management.
- **Prompt Engineering and RAG:** Combining symbolic knowledge with model reasoning.
- **Fine-Tuning:** LoRA, PEFT, quantization for efficient adaptation.
- **Data Handling:** Cleaning and curating domain-specific corpora.

# Difficulties: Infrastructure

- **Compute:** Training is costly; inference still GPU-intensive.
- **Pipelines:** Need to process both structured and unstructured data.
- **Security & Compliance:** Protecting sensitive data.
- **Integration:** Aligning with legacy systems and APIs.

# Framing for Engineers

- Transformers are *not magic*, but a new abstraction for processing information.
- Apply them at company bottlenecks: support tickets, document processing, anomaly detection.
- Start small: prototypes $\rightarrow$ production integration.
- Success requires collaboration: engineers, domain experts, ML specialists.

# Closing Message

- Transformers are becoming **general-purpose engines** for pattern recognition.
- They provide leverage where rules and heuristics fail.
- Adoption is not just about technology but about building:
  - Knowledge
  - Skills
  - Infrastructure
- Goal: make them reliable at enterprise scale.

# Pipeline Goals

- **Relevance:** return truly useful chunks for LLM retrieval.
- **Freshness:** keep indexes up-to-date with changing data.
- **Traceability:** every chunk linked to source + version.
- **Scalability & Latency:** handle growth and meet SLOs.
- **Security:** compliance, audit, access control.
- **Cost-effectiveness:** balance compute vs storage.

# Data Sources

- Documents: PDFs, HTML, Word, Markdown.
- Databases: product DBs, transactional systems (CDC).
- Logs & telemetry: server logs, sensors, time-series.
- Code repositories: Git, issues, PRs.
- Communication: emails, tickets, chat transcripts.
- APIs: third-party SaaS, exports.

# End-to-End Stages

1. Ingestion (batch, streaming, CDC).
2. Extraction & normalization (text cleaning, schema mapping).
3. Deduplication & filtering (hashing, PII removal).
4. Chunking (200–1000 tokens, overlap 50–200).
5. Embedding generation (batching, caching, versioning).
6. Indexing in vector DB (HNSW, hybrid search).
7. Retrieval + optional reranking.
8. Prompt construction + LLM orchestration.
9. Post-processing: source attribution, feedback loop.

# Chunking & Embeddings

- **Chunking:** split into semantically coherent units.
  - Typical size: 512 tokens with overlap.
  - Special rules for code, tables, structured docs.
- **Embeddings:**
  - Batched inference on GPU.
  - Idempotent: embed only once per model+chunk.
  - Versioning: reindex on model upgrades.

# Indexing & Retrieval

- Store vectors + metadata (doc_id, offsets, model version).
- Vector DB: Qdrant, Milvus, Pinecone, Weaviate, Elasticsearch.
- Hybrid retrieval: combine BM25 + vector similarity.
- Rerankers: cross-encoders for higher precision.
- Filters: metadata-based restrictions (region, product, sensitivity).

# Prompt Orchestration

- Assemble top-K chunks into prompt context.
- Manage token budget: system prompt + context + answer.
- Add explicit citations (provenance).
- Fallbacks for low-confidence retrieval.
- Human-in-the-loop for sensitive outputs.

# Operational Concerns

- **Compute:** GPU for embeddings, index scaling.
- **Storage:** cold originals (S3/Blob), hot vectors.
- **Governance:** PII redaction, access control, audit logs.
- **Observability:** monitor ingestion lag, retrieval precision@k, LLM latency.
- **Testing:** regression suites + human evaluation.

# Best Practices

- Start small, iterate: prototype on 1–2 sources.
- Keep originals immutable, always version embeddings.
- Use hybrid retrieval for robustness.
- Constrain LLMs: encourage extractive answers.
- Collect feedback to improve rerankers and retrievers.
- Control costs: batch embeddings, quantize vectors.

# Common Pitfalls

- **Under/over-chunking** $\rightarrow$ poor retrieval.
- **No metadata filtering** $\rightarrow$ irrelevant results.
- **Silent re-embedding** $\rightarrow$ inconsistency.
- **Blind trust in LLMs** $\rightarrow$ hallucinations.
- **Stale indices** $\rightarrow$ outdated answers.

# Quick Checklist

1. Select 1–2 data sources (e.g., tickets + docs).
2. Ingest, clean, and chunk into 512 tokens.
3. Generate embeddings and store with metadata.
4. Index in a vector DB; expose retriever API.
5. Connect retriever $\rightarrow$ prompt $\rightarrow$ LLM.
6. Instrument precision@k, latency, token cost.
7. Iterate with rerankers and user feedback.

# Closing Message

- Data pipelines are the backbone of RAG.
- Success requires **knowledge, skills, and infrastructure**.
- Transformers are powerful, but only as good as the data pipeline behind them.
- Goal: reliable, scalable, compliant enterprise knowledge systems.