Information Retrieval Evolution

INST447 Data Source and Manipulation

Wei Ai

What is Information Retrieval?

- Narrow Definition: Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

  – *Manning, Raghavan, and Schütze*
- (We will use Text Retrieval and Information Retrieval interchangeably for this lecture)

Broader Definitions

- Data can be semi-structured, multi-modal, abstract.
  - Image, video, opinion, expert, …
  - Although in this class, we will only talk about text
- Information need can be implicit, dynamic, inaccurate
  - Information filtering, recommender systems
  - Sometimes there isn't a query!
- Find information → knowledge acquisition
  - Give me what you have → tell me what you know
- Relevance isn't the only criterion.
  - Novelty, diversity, personalization, …

Short vs. Long Term Info Need

- Short-term information need (Ad hoc retrieval)
  - "Temporary need", e.g., info about used cars
  - Information source is relatively static
  - User "pulls" information
  - Application example: library search, Web search
- Long-term information need (Filtering)
  - "Stable need", e.g., new data mining algorithms
  - Information source is dynamic
  - System "pushes" information to user
  - Applications: news filter, recommender systems

Era 1: Boolean Retrieval

- Documents and queries as sets of terms;
- Matching/relevance via set operations: AND/OR/NOT
- Example:
  - `(data AND science) OR (machine AND learning) NOT statistics`
- Fast, interpretable, controllable; fragile to vocabulary mismatch
- Still everywhere (filters, fielded search, grep, SQL LIKE)

Era 1: Boolean Retrieval

```
(
  "Vaccination Hesitancy"[Mesh]
  OR (vaccin*[tiab] AND (hesitan*[tiab] OR accept*[tiab] OR refusal*[tiab] OR uptake[tiab] OR intend*[tiab]))
)
AND
(
  "Health Personnel"[Mesh]
  OR healthcare worker*[tiab] OR health care worker*[tiab] OR "medical staff"[tiab]
  OR nurse*[tiab] OR physician*[tiab] OR clinician*[tiab] OR doctor*[tiab]
)
AND
(
  "Surveys and Questionnaires"[Mesh]
  OR Survey[Publication Type]
  OR survey*[tiab] OR questionnair*[tiab] OR "cross-sectional"[tiab] OR cross sectional[tiab] OR prevalence[tiab]
)
AND humans[Mesh]
NOT animals[Mesh]
NOT (Editorial[ptyp] OR Letter[ptyp] OR Comment[ptyp] OR News[ptyp])
AND english[lang]
AND ("2019/01/01"[dp] : "3000"[dp])
```

Beyond Boolean Retrieval

- Given a query, how do we know if document A is more relevant than B?

- One Possible Answer:
  If document A uses more query words than document B
  (Word usage in document A is more similar to that in query)

Revisit: Vector Representation and Similarity

- Represent a doc/query by a term vector
  - Term: basic concept, e.g., word or phrase
  - Each term defines one dimension
  - N terms define a high-dimensional space
  - Element of vector corresponds to term weight
  - E.g., $d = (x_1, \ldots, x_N)$, x_i is "importance" of term i
- Measure relevance by the similarity/distance (e.g., the cosine similarity) between the query vector and document vector in the vector space

What the VS model doesn't say

- How to define/select the "basic concept"
  - How to select index terms
  - Concepts are assumed to be orthogonal
- How to assign weights
  - Weight in query indicates importance of term
  - Weight in doc indicates how well the term characterizes the document
  - We talked about simple presence/absence
- How to define the similarity/distance measure

Era 2(a): Vector Space (Sparse Vectors)

- Key techniques:
  - Vector Space Model: Documents and queries as vectors
    - Rank documents by vector similarity
  - TF-IDF: Term frequency × Inverse document frequency
    - How important is a word to *this* doc?
- Relevance = Similarity

Term Frequency (TF) Weighting

- Idea: A term is more important if it occurs more frequently in a document
- Formulas:
  - c(t,d): the frequency count of term t in doc d
  - Raw TF: TF(t, d) = c(t, d)
- We always need to normalize the raw TF
  - “Repeated occurrences” are less informative than the “first occurrence”
  - Log TF: TF(t, d) = log(c(t, d) + 1)

Inverted Document Frequency (IDF) Weighting

- Idea: A term is more discriminative if it only occurs in fewer documents.
  - Why is this true?
- Fomula:
  - $IDF(t) = 1 + \log(\frac{n}{k})$
  - N - total number of documents in collection
  - K - number of document with term t (document frequency)

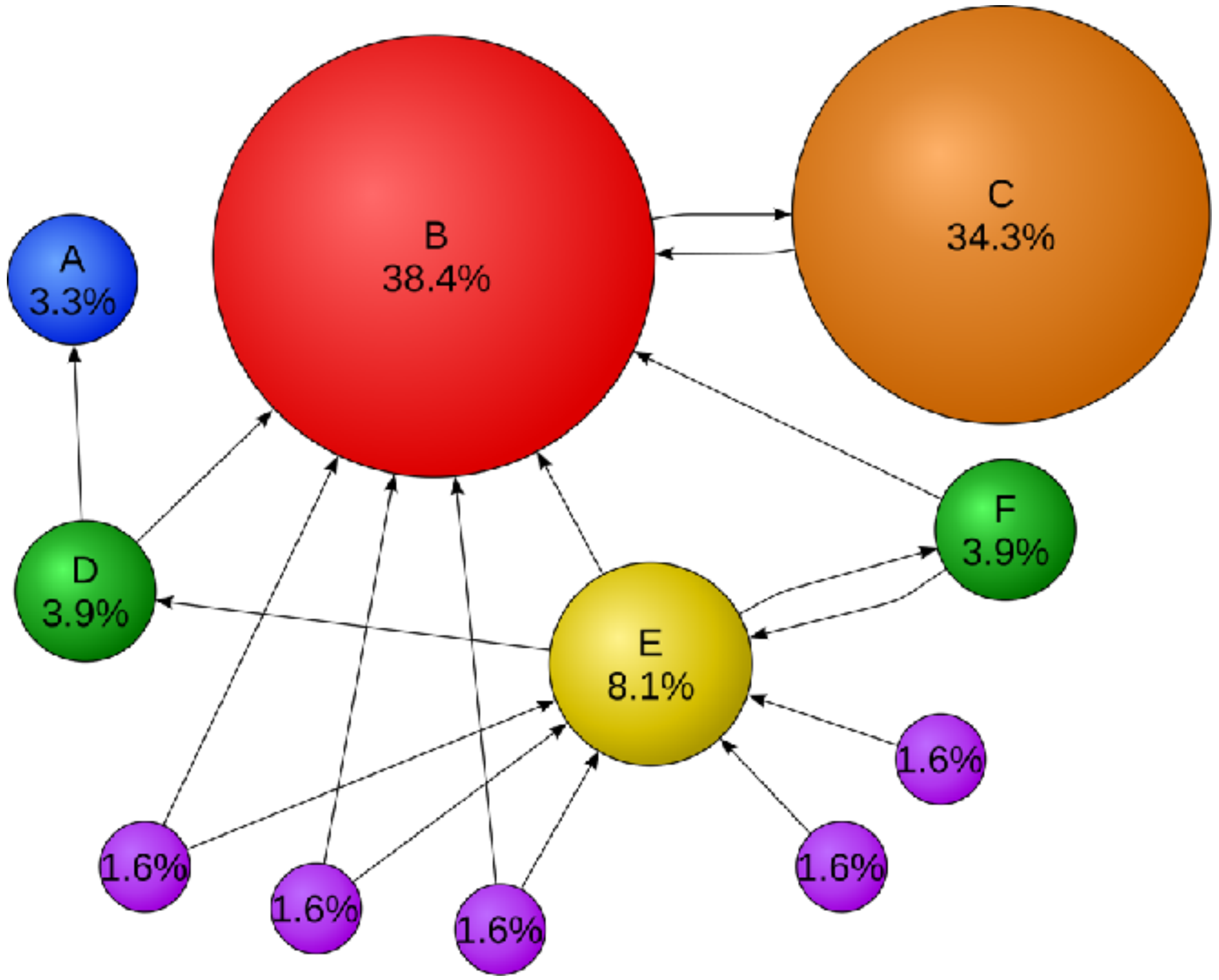
Note that IDF is document independent while TF is document dependent!

Era 2(b): Probabilistic Models

- "What's the *probability* this doc is relevant?"
- Relevance = Probability
- BM25 - the “king” of baselines
  - A "bag of words" model, but smarter:
  - it accounts for term saturation (the 10th "data" is less important than the 1st)
  - and document length — penalizing longer document.

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

Era 3: Graph Signals



- Vector similarity isn't enough
  - "Best Python tutorial" returns random blog posts
  - Need to know which sources are authoritative

- PageRank and variants: use the web graph to estimate authority

- The Pipeline
  - Retrieve: Use vector methods (BM25) to find candidate documents
  - Rank: Combine text relevance + PageRank score
  - Present: Top results balance relevance AND authority

Era 4: Learning to Rank (LTR)

- Why use just one or two signals? Let's use hundreds and “learn" the best ranking formula.
- Assembling multiple features:
  - BM25 scores, PageRank score, user behavior, freshness, …
- Pointwise/pairwise/listwise training with editorial/click data
- Classical: RankSVM, Gradient Boosted Trees (LambdaMART)
- Often used as re-ranker over a first-pass retrieve.

Era 5: Neural Information Retrieval

- From Sparse to Dense
- The shift: Vectors become embeddings
  - **Sparse Vector**: 50,000+ dimensions, full of zeros. [0, 0, 1, 0, ... , 0, 2, 0]
  - **Dense Vector**: 300-1000+ dimensions, all have meaning.[0.23, -0.45, 1.03, ... , 0.91]
    - Capture semantic similarity: "car" close to "automobile"
- Key technologies:
  - Word2Vec, GloVe (word embeddings)
  - BERT, RoBERTa (contextual embeddings)
  - Sentence transformers (document embeddings)
- Retrieval function is now a deep neural network). You find documents that are **semantically** close in this **embedding space**, even if not sharing keywords.

Vector Database

- Traditional Databases are built for exact matches and filtering on structured data.
  - e.g. `SELECT * FROM docs WHERE author = 'Smith'`
  - Terrible at finding "nearest neighbors" in 1000-dimensional space. A `WHERE` clause can't do that.
- Problem: 10M documents × 768 dimensions = 7.68 billion numbers
  - Can't do linear search (too slow!)
  - Traditional SQL databases are terrible at finding "nearest neighbors" in such dense vector space.
- Solution: Vector Databases (Pinecone, Chroma, FAISS)
  - Use approximate nearest neighbor search (ANN)
  - Trade perfect accuracy for speed

RAG — Retrieval-Augmented Generation

- The pipeline:
  - Retrieve: Find relevant documents (using methods we discussed)
    - Query understanding: expand/rewrite, detect intent
    - Retrieve: BM25/sparse, dense, or hybrid
    - Re-rank: cross-encoder for top-k quality
  - Augment: Add retrieved docs to LLM prompt
  - Generate: LLM synthesizes answer with citations
    - Post-process: guardrails, citation verification, caching
- Why RAG?
  - Ground LLMs with external knowledge,
  - reduce hallucination, add freshness.
  - Allows dynamic knowledge updates

The Agentic AI Twist

```
grep -r "TODO" ./src
find . -name "*.py" -exec grep "def process_data" {} \;
```

- LLM agents often skip RAG, or even dense vectors.
- Observation: Claude Code, Devin, Cursor use grep and shell commands
- Why?
  - Speed: grep is instant, no GPU needed
  - Structured data: Code has structure (function names, file paths)
  - Exact matching: When you want "error code 404", not "similar to error"
  - Transparency: Can explain what was searched
- Different data genres need different retrieval methods!
  - Unstructured text → embeddings (semantic search)
  - Structured code → regex/grep (exact match)
  - Logs → SQL (structured queries)
  - Knowledge graphs → graph traversal
- Know your data type, choose your representation!

Long-Context LLMs: IR Killer?

- The dream: 1M+ token context
  - Dump all documents into context
  - Ask questions
  - No retrieval needed!
- The reality check:
  - "Needle in haystack" problem: LLMs still miss info in long contexts
  - Cost: Processing 1M tokens is expensive
  - Latency: Takes time to process
  - The core IR problem remains: Users still need help finding what matters

Long Context ≠ No Retrieval

- Even with infinite context...
- You still need IR for:
  - Filtering: What goes into the context?
  - Prioritization: What appears first? (LLMs attend to start/end more)
  - Summarization: Can't show user 1M tokens
  - Cost optimization: Only retrieve what's needed
- The constant: Understanding and satisfying information needs without overwhelming users.

Evaluation: What to Measure (and Why)

- Search is not classification
- Scenario: Search for "machine learning tutorials"
  - 1 million relevant documents exist
  - System returns 10 documents
- Questions:
  - Did we find good ones? → Precision
  - Did we find all of them? → Recall
  - Are the best ones at the top? → NDCG, MAP
- Other practical metrics:
  - Diversity/novelty, coverage; latency; cost

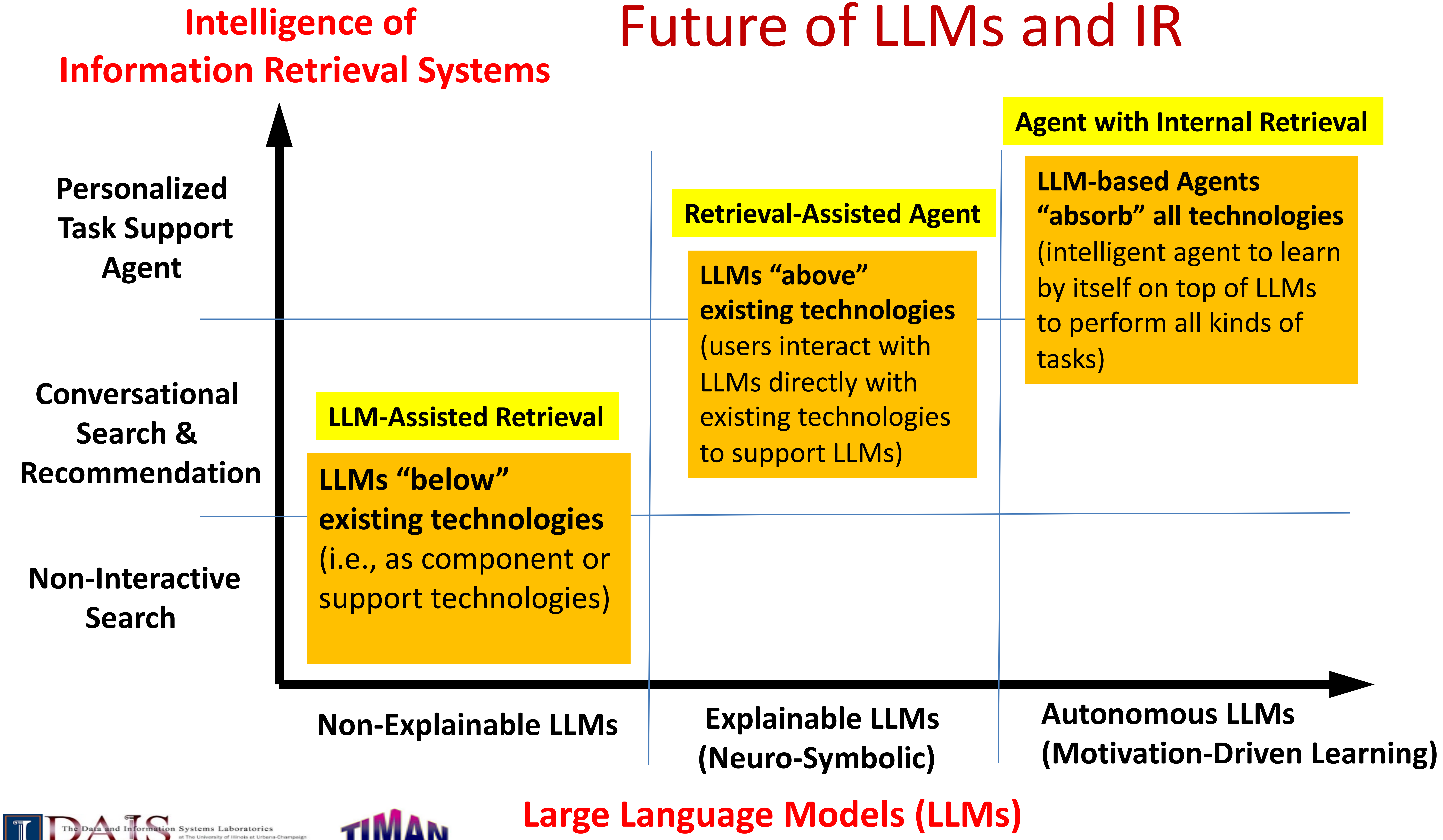
Evaluation in the LLM Era

- New metrics for new tasks.
- Classical IR: Rank documents
- LLM + RAG: Generate answers
  - Faithfulness: Did LLM stick to retrieved docs?
  - Attribution: Can we trace claims to sources?
  - Answer quality: Is it correct, complete, concise?
- The evolution: Task changed, metrics evolved, but still measuring "information need satisfaction"

Looking Ahead: IR to serve humans and AI agents

- IR to serve human users: from search engines to task agents
  - Human-centered design
  - Information seeking behavior / intent modeling / preference learning…
- IR to serve AI agents: five essential new IR tasks
  - LLM for IR agents (optimize the pipeline)
  - Technical: indexing, retrieval models, re-ranking, vector DB ops, evaluation, …

Future of LLMs and IR



Safety, Privacy, and Integrity

- Prompt injection & data exfiltration in RAG/agents
- Copyright & licensing of indexed sources
- PII/PHI handling; on-device retrieval, encryption, access control
- Guardrails: allowlists, input/output filters, sandboxed tools,…

When Working on IR Problems

- Ask these questions:
  - What is the data type? (text, code, logs, structured)
  - What is the scale? (1K docs vs 1B docs)
  - What is the need? (exact match vs semantic search)
  - What are the constraints? (latency, cost, accuracy)
- Then choose:
  - Exact string matching (grep, SQL)
  - Sparse retrieval (BM25)
  - Dense retrieval (embeddings)
  - Graph-based (PageRank)
  - Hybrid (usually the answer!)

Case Study: Build a Health/Medical Search Engine

Question	Answer	Implication
Data type?	Scientific text + structured drug data	Need both semantic understanding AND exact matching
Scale?	34M+ documents	Need fast first-pass filtering
User need?	Doctors: precise terms / Patients: natural language	Must serve different query styles
Constraints?	Accuracy critical, must explain results	False info = harm; need source attribution

Case Study: Build a Corporate Knowledge Base

Question	Answer	Implication
Data type?	Text docs + code + messages + structured tickets	Different sources need different methods
Scale?	100K-1M items, 1000+ updates/day	Moderate scale, high freshness need
User need?	Known-item ("Q3 report") + exploratory ("our AI work")	Mix of exact and semantic
Constraints?	**Privacy critical**, cost-conscious, <500ms latency	Permissions first! Budget matters

Takeaway

- Information Retrieval is about understanding and meeting information needs.
- The representations change
  - sets → sparse vectors → dense vectors → ?
  - graph, stream, …
- The core mission stays
  - find the right information, don't overwhelm
- Your role as data professionals:
  - Choose the right tool for the right job,
  - understand the tradeoffs,
  - and keep learning!

If You Want to Build Your Own IR System

- Classic IR:
  - Elasticsearch: Industry standard
  - Apache Solr: Open source alternative
- RAG Frameworks:
  - LangChain: Full-featured, lots of integrations
  - LlamaIndex: Focused on RAG
- Vector Databases (free tiers available):
  - Chroma: Local, easy to start
  - Pinecone: Cloud, generous free tier
- Evaluation:
  - TREC datasets
- Start small, iterate, learn by building!