


Designing Machine Learning Systems

An Iterative Process
for Production-Ready
Applications



Chip Huyen

- * Read a chapter a week
- * Livestream discussions every Saturday  [@MLOpsLearners](#)
- * Starts Saturday April 13th 2024!

Chapter 3: Data Engineering Fundamentals

Group 7 (Team Q*)

Saturday, April 27, 2024



https://www.youtube.com/watch?v=2ryiXysW6_c

Chip Huyen's Designing ML Systems Book Club

Data Engineering Fundamentals

Cohort 1, Group 7 "Team Q*"

April 27th, 2024

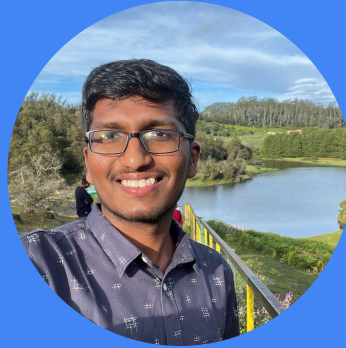
The team: Q*



Nikhil Sharma

CS @ YorkU

[Let's Connect](#)



Vineeth Kada

ML @ CMU
Next @ DatologyAI

[Let's Connect](#)



Krishna Maddula

Building ChatBots &
Agents @ Amex

[Let's Connect](#)



Om Amitesh

CS @ UCLA

[Let's Connect](#)

Agenda

Chapter Summary/ Key takeaways

What surprised you about the chapter?

Related Examples & Case Studies

Data Sources

- Various sources of Data can be Users, Systems, User events resulting in System data, Internal Databases
- Data can come in various systems (First Party/Second Party/Third Party)
- Rise of Internet and smartphones made it easy to collect data.
- Data of all kinds can be bought.
- As demand for data privacy increasing, companies like Apple have been taking steps to curb usage of advertiser IDs by making IDFA (Apple's Identifier for Advertisers) Opt-in

Data Formats

- Multimodal data?
- Where to store? Cheap and still fast to access? Human readable?
- How to store models?
- Persistent / Impersistent?

- CSV, Numpy - Row major
- Parquet, Pandas - Column major
- CPU - Row major favors spatial locality (due to caching)

Table 3-1. Common data formats and where they are used

Format	Binary/Text	Human-readable	Example use cases
JSON	Text	Yes	Everywhere
CSV	Text	Yes	Everywhere
Parquet	Binary	No	Hadoop, Amazon Redshift
Avro	Binary primary	No	Hadoop
Protobuf	Binary primary	No	Google, TensorFlow (TFRecord)
Pickle	Binary	No	Python, PyTorch serialization

Row Major vs. Column Major in GPU

AoS versus SoA

(Row Major)

```
struct node {  
    int a;  
    double b;  
    char c;  
};  
struct node allnodes[N];
```

Expectation: When a thread accesses an attribute of a node, it also accesses *other attributes* of the *same node*.

Better locality (on CPU).

(Column Major)

```
struct node {  
    int alla[N];  
    double allb[N];  
    char allc[N];  
};
```

Expectation: When a thread accesses an attribute of a node, its *neighboring thread* accesses the *same attribute* of the *next node*.

Better coalescing (on GPU).

- Takeaway

- CPU - Array of Structures
- GPU - Structure of Arrays

GPU = SIMD

Single Instruction Multi Data

EXAMPLE

marks = student[i].marks

student[i].grade = computeGrade(marks)

CPU - st[1].m, st[1].g, st[2].m, st[2].g

GPU - (st[1].m, st[2].m); (st[1].g, st[2].g)

() = In parallel

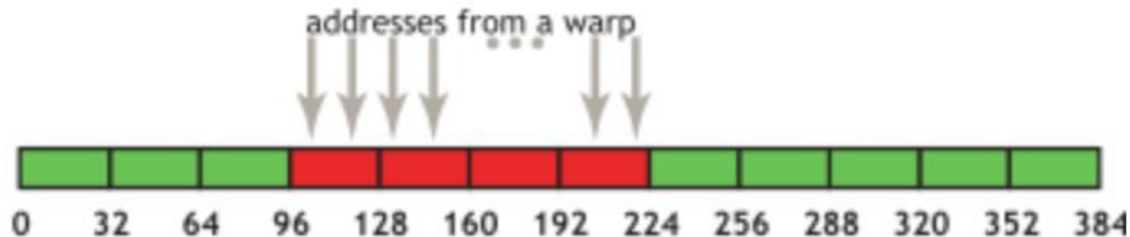
9.2.1. Coalesced Access to Global Memory

A very important performance consideration in programming for CUDA-capable GPU architectures is the coalescing of global memory accesses. Global memory loads and stores by threads of a warp are coalesced by the device into as few as possible transactions.

Note

High Priority: Ensure global memory accesses are coalesced whenever possible.

Four
32 byte
segments



Coalesced access

Text Vs Non-Textual

- Binary allows further compression
 - Like Huffman encoding
 - Numbers - 12341234 (text - 8 bytes, int32 - 4 bytes)
- BPE (used in LLMs like GPT)
 - Also does some sort of compression
 - It is the most efficient way to encode the given corpus with the specific number of tokens.
 - Learn more: <https://huggingface.co/learn/nlp-course/en/chapter6/5>

Char ♦	Freq ♦	Code ♦
space	7	111
a	4	010
e	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
l	1	11001
o	1	00110
p	1	10011
r	1	11000
u	1	00111
x	1	10010

Huffman Encoding

Data Models (Relational vs. Non)

- **Structure:**
 - Relational: Structured data in tables with rows and columns.
 - Document: JSON-like documents with dynamic schemas.
 - Graph: Nodes, edges, and properties to represent and store data.
- **Schema Flexibility:**
 - Relational: **Fixed schema**; modifications require downtime.
 - Document: **Flexible schema**; easy to evolve data model.
 - Graph: Highly **flexible**; suitable for interconnected data.
- **Use Cases:**
 - Relational: Banking, Inventory, HR systems.
 - Document: Content management, Mobile apps, Real-time analytics.
 - Graph: Social networks, Fraud detection, Recommendation engines.
- **Scalability:**
 - Relational: Vertical scaling; scale by increasing hardware.
 - Document & Graph: Horizontal scaling; distribute data across more servers.
- **Query Capability:**
 - Relational: Powerful SQL queries; complex joins.
 - Document: Query based on document content; simpler queries.
 - Graph: Efficient for complex, deep relationships.

Non Relational Models

Careful where you draw the line for document models



ChatGPT

"Kicked off with plain text files to avoid the SQL maze. First, added indexes to speed things up. Next, structured our data for faster access. Then, threw in some joins for fun. Several 'optimizations' later, we proudly reinvented our own wheel—SQL! Who knew? 🔄🤖"

Entity-Relationship (ER) Model

Table 3-2. Initial Book relation

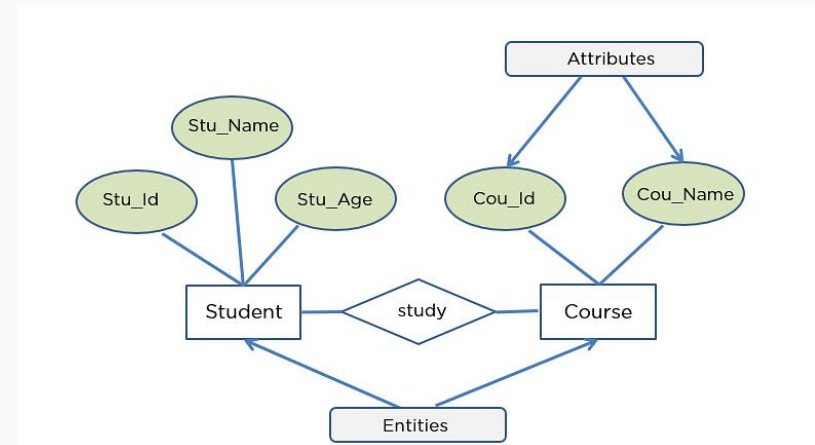
Title	Author	Format	Publisher	Country	Price
Harry Potter	J.K. Rowling	Paperback	Banana Press	UK	\$20
Harry Potter	J.K. Rowling	E-book	Banana Press	UK	\$10
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	\$30
The Hobbit	J.R.R. Tolkien	Paperback	Banana Press	UK	\$30
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	\$15

Table 3-3. Updated Book relation

Title	Author	Format	Publisher ID	Price
Harry Potter	J.K. Rowling	Paperback	1	\$20
Harry Potter	J.K. Rowling	E-book	1	\$10
Sherlock Holmes	Conan Doyle	Paperback	2	\$30
The Hobbit	J.R.R. Tolkien	Paperback	1	\$30
Sherlock Holmes	Conan Doyle	Paperback	2	\$15

Table 3-4. Publisher relation

Publisher ID	Publisher	Country
1	Banana Press	UK
2	Guava Press	US



Entity-Relationship (ER) Model

- Reduces redundancy
 - University has students with attributes Dept, Year, DOB etc.
 - Let's say we have an attendance system
 - Everytime a student logs in if we enter all the attributes it's waste of time
 - So, assign an ID and store the attributes some place else which rarely changes

Relational Databases

- Query languages
 - SQL - the most popular
 - Declarative - we just say what we want not how to do it!
 - One application of ML for databases
 - ML can be used in query optimizers
 - Optimizer essentially has to find the most efficient way to execute the query

Non Relational Models

Downsides of relational models

- Restrictive - strict schema
- Writing and executing SQL queries

NoSQL = **Not only SQL** (can support relational too)

Non Relational Models

- Document model
 - **Simple** - set of documents
 - **Flexible** - can be modified to take advantage of access patterns
 - Put all info about a book in a document instead of splitting it across tables
- Graph model
 - Nodes - data; Edges - relations

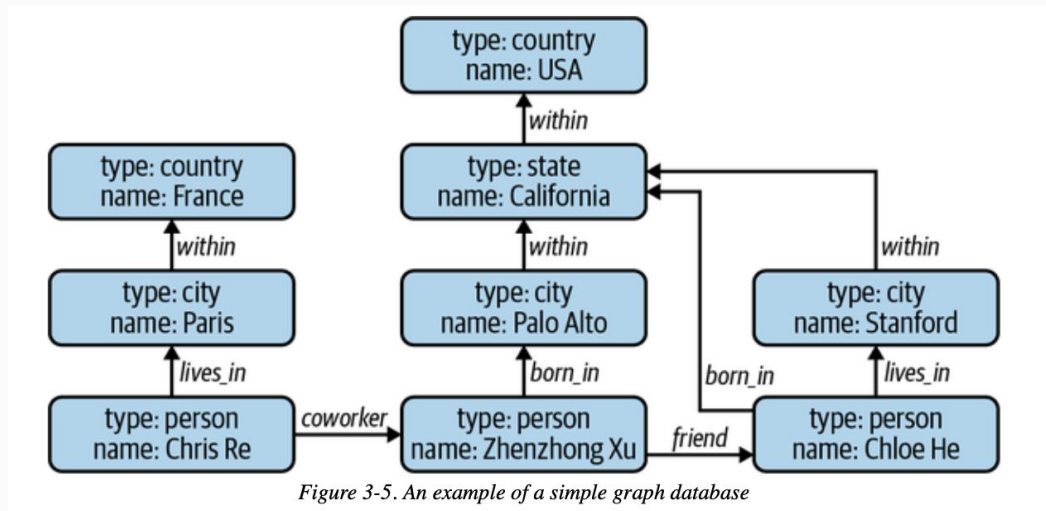


Figure 3-5. An example of a simple graph database

Data Storage Engines and Processing

- Storage Engines are basically databases, they define how we:
 - Store Data
 - Retrieve Data
- Two types of Workloads that Databases are (were?) optimized for:
 - Transactional Processing
 - Analytical Processing



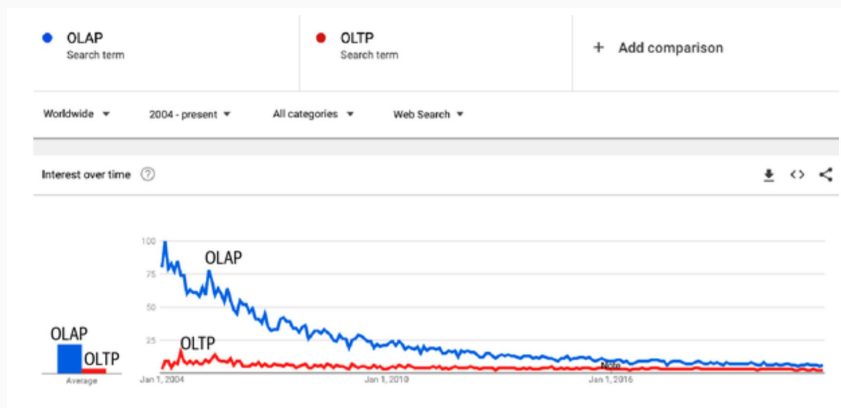
OLTP and OLAP

	OLTP	OLAP
Workflow	<ul style="list-style-type: none">• Capture Activity as 'Transactions'• Inserted-As-Generated• Occasional Updates• Deleted when not needed	<ul style="list-style-type: none">• Quick Multidimensional Reports• Derive Trends and Insights• Long Term Data Retention
Prioritize	<ul style="list-style-type: none">• Ticking all ACID properties would be ideal• But, BASE also works	<ul style="list-style-type: none">• Build Structures that enable different viewpoints• Aggregations, Aggregations, Aggregations!

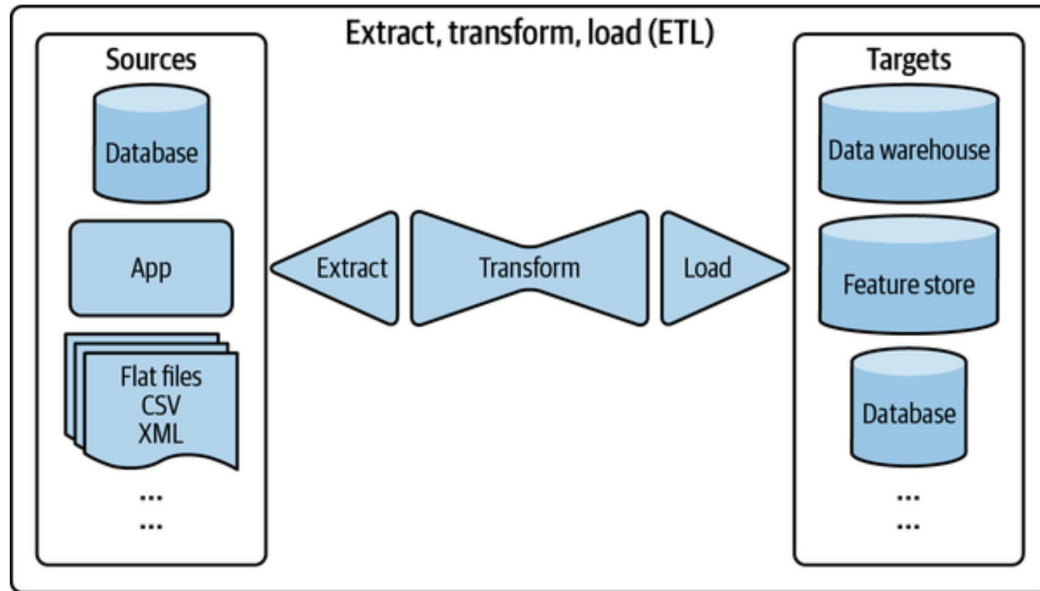
OLAP and OLTP: Outdated?

Yep, that's right. Outdated. Why? :

- Technology is no longer limited:
 - Transactional DBs with analytical powers: CockroachDB
 - Analytical DBs with transactional powers: Apache Iceberg, DuckDB
- Storage - Processing Decoupling:
 - Storage layer and Processing layer independence
- 'Online' has different meaning
 - *Online* before: connected to internet/in prod
 - *Online* now is data processing speed - online, nearline, offline.

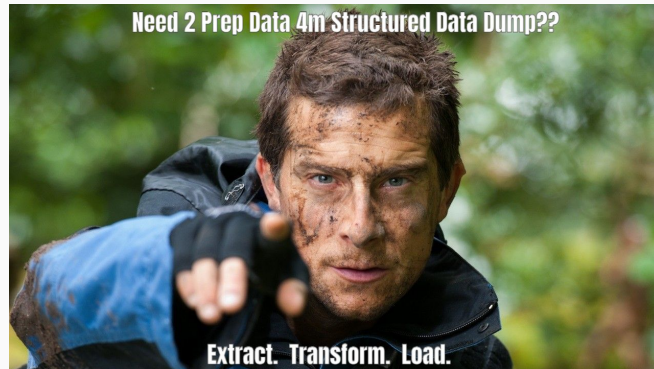


ETL: Extract. Transform. Load.

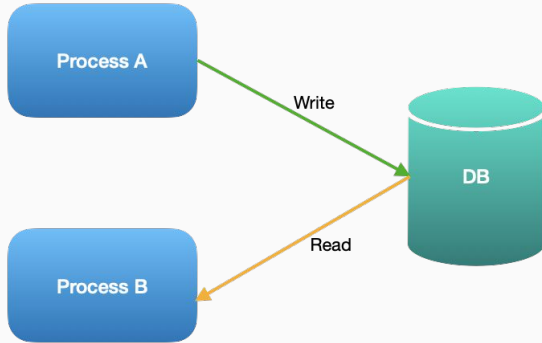


ETL: When to?

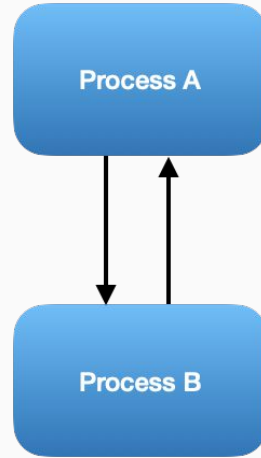
- ETL makes sense when you have a dump of structured data you can *readily* filter out.
- If you don't want to keep data structured, the idea of ELT - Extract, Load, Transform makes it faster
- Companies are now shifting from ELT to ETL
- Data Vendors today give you hybrid options with Data Lakehouses!



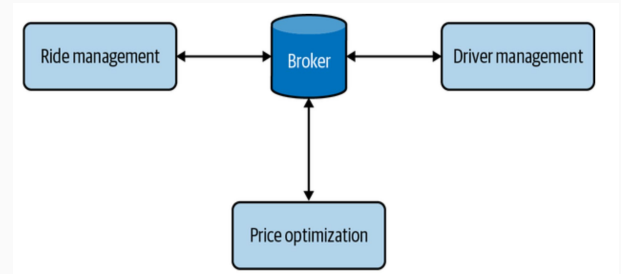
Modes of Dataflow



Passing through DB



Passing through Services



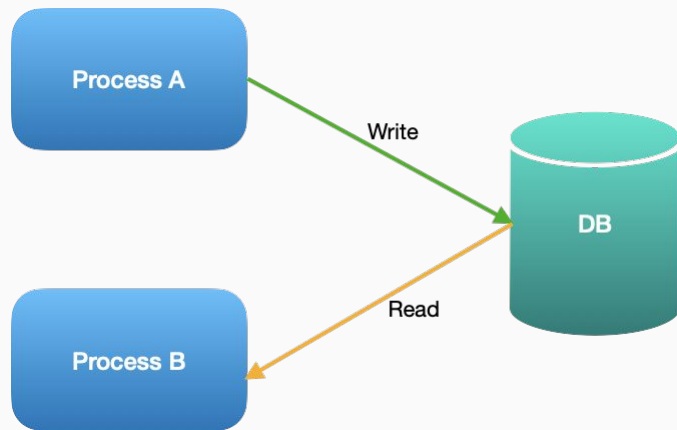
Passing through Real-Time transport

Modes of Dataflow: Key Takeaways

- Data Passing through a DB is a bad idea for most consumer facing apps - due to read/write latency
- Data Passing through Services works and ensures decoupling of processes but fails when there are a 1000 processes - likely for ML services requesting *features*
- Data Passing through a Real-Time transport works the best for data-intensive applications (Kleppmann approved!) - But might be overkill for small scale

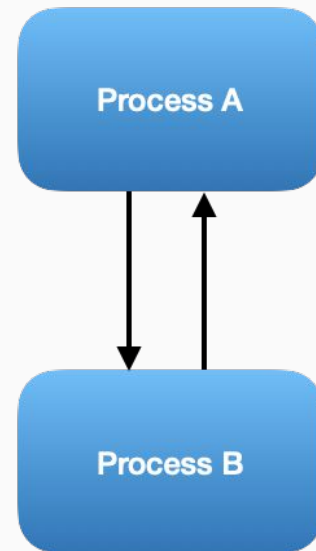
Data passing through Databases

- Simple Idea -
 - Process A writes to **DB**
 - Process B reads from same **DB**
- Good Idea? Nope. Why? :
 - If processes exist on different companies, they don't share DBs!
 - DBs have read/write latency! - not suitable for consumer facing apps!



Data Passing through Services

- Each Process is viewed as a service to one-another
- If Process A wants some data from Process B:
 - Process A sends a request to Process B for what it wants
 - Process B uses the request to get Process A what it wants
 - Process B sends a response to Process A through the same network
- Good Idea? Yep.
 - Processes On different Companies? Handled, we use a network - based solution
 - Processes are separate - They're developed, tested and maintained independently!



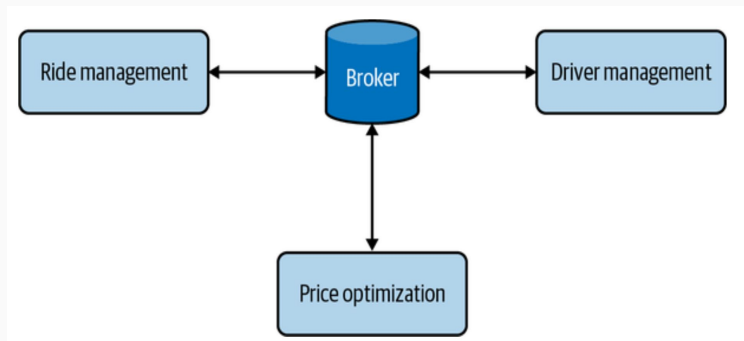
Data Passing through Services: Issues?

- Data passing gets complicated with 1000s of processes coming in!
- Network starts becoming a bottleneck and can slow us down :(
- A dead/failed process in production could lead to all dependent processes down :/



Data passing through real-time transport

- Processes talk to a data passing broker
- Data broadcast to the broker is called an event
 - brokers are event-driven
- The Broker acts like an in-memory storage for the latest information from each service
- Any process needing info, simply asks the broker
- Good Idea? Yeah:
 - Works better for data-heavy applications
 - Decoupling - services that produce data don't care about the consumers



Batch processing Vs Stream processing: Key Takeaways

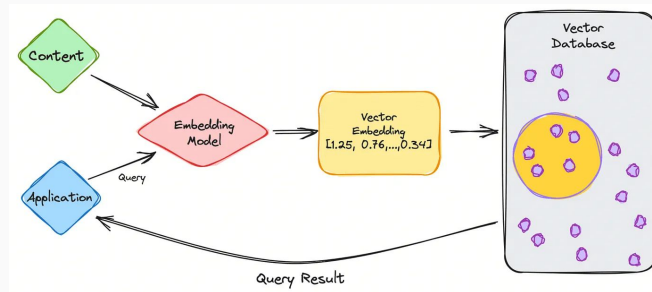
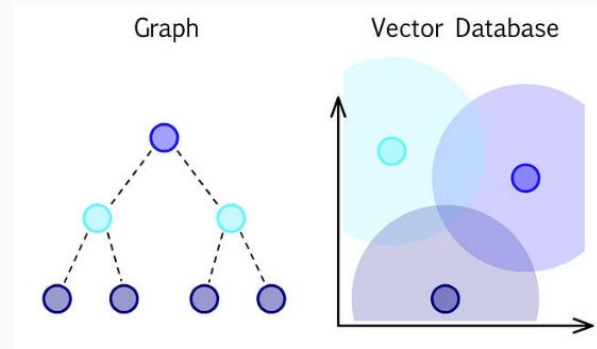
- Batch processing happens periodically - we wait for data to arrive in our storage engine and then kick off processes
- Stream processing is computation on real-time sources
- In ML Systems :
 - Batch processing is used to compute features that change less often - static features
 - Stream processing is used to compute features that change quickly - dynamic features
- Stream processing is more difficult for ML systems
- Stream processing is more difficult due to variable speed and undefined size

What's changed and same with GenAI?



The Rise of Specialized Databases

- Traditional databases struggle with AI workloads, leading to the emergence of specialized databases.
- VectorDBs: optimized for similarity search, clustering, and nearest neighbor queries. Examples: FAISS, Pinecone, Qdrant, Milvus
- GraphDBs: designed for storing and querying complex relationships between data entities. Examples: Neo4j, Amazon Neptune
- These databases are becoming mainstream, driven by the growth of AI applications.

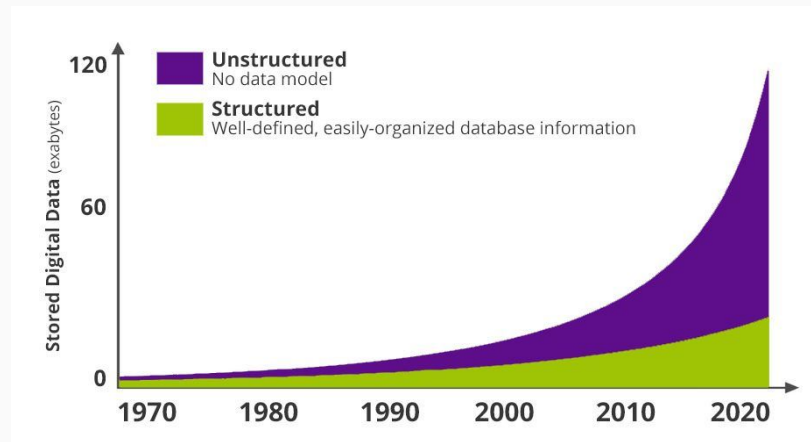


The Impact of GenAI on the Data Landscape

- Generative AI (GenAI) is revolutionizing data creation, processing, and analysis.
- GenAI models generate vast amounts of data, leading to an explosion in data volume and velocity.
- Traditional data management systems struggle to keep up with the scale and complexity of GenAI-generated data.
- New data management architectures and technologies are emerging to address these challenges.

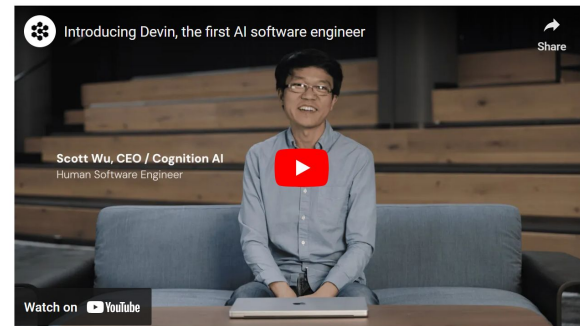
Unstructured data and LLMs

- Unstructured data (e.g., images, videos, text) is becoming increasingly important in GenAI applications.
- Large Language Models (LLMs) are driving the growth of unstructured data, with models like BERT and transformer-based architectures.
- Unstructured data requires specialized storage, processing, and analysis techniques.
- LLMs are pushing the boundaries of traditional data management systems, driving the need for new architectures and technologies.



GenAI Use Cases: Transforming Industries

- **Banking**: Create personalized financial planning tools that generate customized investment portfolios based on their risk tolerance and financial goals.
- **Healthcare**: Chatbots that generate personalized responses to patients' queries, improving patient engagement and experience.
- **Education**: Create personalized learning paths that generate customized lesson plans, exercises, and quizzes tailored to individual students' learning styles and abilities.
- **Tech**: Develop AI-powered coding assistants that generates code, automate programming tasks and tests, and even write entire programs from scratch.



March 12th, 2024 | Written by Scott Wu

Introducing Devin, the first AI software engineer

Data Platforms for GenAI

- Traditional data platforms are not designed to handle the scale and complexity of GenAI workloads.
- New data platforms are emerging to support GenAI, offering features like:
 - a. Scalability and performance
 - b. Real-time data processing and analysis
 - c. Support for specialized databases (e.g., VectorDBs, GraphDBs)
 - d. Integration with AI frameworks and libraries (e.g., TensorFlow, PyTorch)
- Examples of GenAI-focused data platforms: Databricks, Apache Spark, Google Vertex AI

In your breakout rooms...

1. For each the following use cases, discuss the features needed. How fresh do we need each of the features to be? Levels of freshness to consider: <100ms, 10m, 1h.
 1. Transaction (credit) fraud detection
 2. Churn prediction
 3. Customer support chatbots
 4. Online price optimization (e.g. for Amazon)
2. How AI can change the way we store and discover data?
3. Do we need specialized vector databases or can vector search be a feature of existing databases?
4. Wes McKinney's: "my rule of thumb for pandas is that you should have 5 to 10 times as much RAM as the size of your dataset." Why does pandas require so much memory? Why is pandas still so common given that it has so many flaws and there are many better alternatives?
5. Given a stream of data continually coming in (you don't know what the next value will be like) and you don't have enough memory to store all the values, how do you compute the median of this data stream with each new value?

Thanks

[Bookclub Repo](#)