COURSE NAME

LECTURE 01 – PROJECT

SUBTITLE

Niklas Abraham

DD Month YYYY

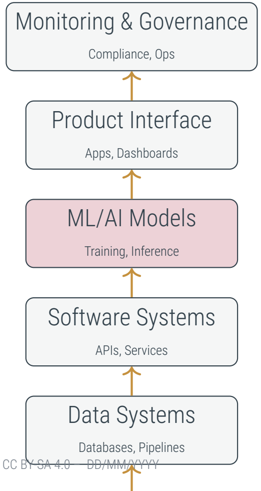
INTELLIGENT FUTURES
AI CONSULTANCY
GLOBAL SOLUTIONS

FACULTY NAME
DEPARTMENT NAME
CHAIR NAME

The Technology Stack: AI in Context

Understanding where AI fits in your organization

AI is not magic—it's software built on data, running on infrastructure, serving business processes.



Executive Reality:

- ▶ AI requires all layers working
- ▶ Model is often < 20% of effort
- ▶ Data quality gates success
- ▶ Governance is not optional

Programming Languages: The Landscape

Why language choice matters for your AI initiatives

Different languages serve different purposes. Understanding this helps evaluate team composition and vendor choices.

Data & ML Ecosystem:

- Python – dominant for ML/AI
  - Rich libraries (TensorFlow, PyTorch)
  - Rapid prototyping
  - Data science standard
- R / MATLAB – statistical analysis niches

Enterprise & Backend:

- Java – enterprise systems, stability
- Go – cloud infrastructure, concurrency
- C# – Microsoft ecosystem

Performance-Critical:

- C/C++ – model runtimes, systems
- Rust – safety + performance
- CUDA – GPU programming

Product Interfaces:

- JavaScript/TypeScript – web, full-stack
- Swift/Kotlin – mobile apps

Specialized:

- Haskell/Scala – type safety, correctness
- SQL – data querying (ubiquitous)

Why Language Choice Matters for AI

Ecosystems, talent, and AI assistance quality

The "Gravity Well" Effect:

- ML research concentrates in Python
- Enterprise gravity in Java/Go
- Performance work in C++/Rust
- Each ecosystem has its own:
  - Package libraries
  - Community expertise
  - Hiring pool

AI Coding Assistant Quality

LLM coding tools perform best where training data is abundant.

Strong support: Python, JavaScript, Java, Go

Moderate: C++, Rust, TypeScript

Weaker: MATLAB, R, niche languages

Executive takeaway: Language choice shapes experimentation speed, maintainability, hiring, and AI-assistance leverage.

Code Comparison: The Same Task in Three Languages

Task: Load CSV, compute summary, detect anomalies, output JSON

Seeing the same logic expressed differently reveals language philosophies.

Python — Concise, library-rich

```python
import pandas as pd
import json

# Load and analyze
df = pd.read_csv("transactions.csv")
summary = {
    "total": df["amount"].sum(),
    "mean": df["amount"].mean(),
    "count": len(df)
}

# Detect anomalies (simple rule)
threshold = summary["mean"] * 3
anomalies = df[df["amount"] > threshold]
summary["anomalies"] = len(anomalies)

# Output
with open("report.json", "w") as f:
    json.dump(summary, f)
```

Characteristics:

- 15 lines of code
- Rich standard library
- Readable, minimal boilerplate
- Dynamic typing (flexible)
- Dominant in data science

Trade-offs:

- Slower runtime than compiled
- Type errors found at runtime
- GIL limits parallelism

Code Comparison: Java – Enterprise Standard

Same task: More structure, explicit types, verbose

Java – Explicit, structured

```java
public class TransactionAnalyzer {
    public static void main(String[] args) {
        List<Transaction> txns = loadCSV("transactions.csv");

        double total = txns.stream()
            .mapToDouble(Transaction::getAmount)
            .sum();
        double mean = total / txns.size();
        double threshold = mean * 3;

        long anomalyCount = txns.stream()
            .filter(t -> t.getAmount() > threshold)
            .count();

        Summary summary = new Summary(
            total, mean, txns.size(), anomalyCount);

        ObjectMapper mapper = new ObjectMapper();
        mapper.writeValue(
            new File("report.json"), summary);
    }
}
```

Characteristics:

- 25 lines (plus class definitions)
- Static typing (compile-time safety)
- Explicit structure
- Enterprise conventions
- Long-lived, maintainable codebases

Trade-offs:

- More boilerplate
- Slower iteration
- Steeper learning curve

Code Comparison: Go — Modern Systems Language

Same task: Explicit error handling, built for services

Go — Explicit, concurrent-ready

```go
func analyzeTransactions() error {
    file, err := os.Open("transactions.csv")
    if err != nil {
        return fmt.Errorf("open: %w", err)
    }
    defer file.Close()

    txns, err := parseCSV(file)
    if err != nil {
        return fmt.Errorf("parse: %w", err)
    }

    var total float64
    for _, t := range txns {
        total += t.Amount
    }
    mean := total / float64(len(txns))
    threshold := mean * 3

    var anomalies int
    for _, t := range txns {
        if t.Amount > threshold {
            anomalies++
        }
    }

    summary := Summary{Total: total, Mean: mean,
        Count: len(txns), Anomalies: anomalies}
    return writeJSON("report.json", summary)
```

Characteristics:

- 30 lines
- Explicit error handling
- Compiled, fast execution
- Built-in concurrency
- Cloud/DevOps standard

Go Philosophy:

- "Errors are values"
- Simplicity over cleverness
- Designed for services

Used by: Docker, Kubernetes, most cloud infrastructure

Development Tools: Editors and AI Coding Assistants

The delivery vehicle for AI in engineering

Modern editors are where AI meets developers—and where governance matters most.

Popular Development Environments:

- VS Code – dominant, extensible, free
- Cursor – AI-native fork of VS Code
- JetBrains (IntelliJ, PyCharm) – enterprise, refactoring
- Vim/Neovim – power users, servers

AI Coding Assistant Capabilities:

- Code completion (line/block)
- Refactoring suggestions
- Test generation
- Documentation writing
- Code search and explanation

Governance Implications

AI coding tools require explicit policies:

- Secrets – API keys, credentials exposure
- IP/Licensing – training data, code ownership
- Security – vulnerable code suggestions
- Auditability – who wrote what?
- Data residency – where does code go?

Database Systems: Choosing the Right Tool

Different data patterns require different systems

"Which database?" is really "What are your query patterns and constraints?"

Relational (SQL):

- PostgreSQL, MySQL, Oracle
- Transactions, consistency, reporting
- Structured business data
- Most enterprise use cases

Document Stores:

- MongoDB, CouchDB
- Flexible schemas
- Product data, events, logs

Key-Value Stores:

- Redis, DynamoDB
- Caching, session state
- Sub-millisecond latency

Columnar / Data Warehouses:

- Snowflake, BigQuery, Redshift
- Analytics at scale
- Historical analysis, BI

Graph Databases:

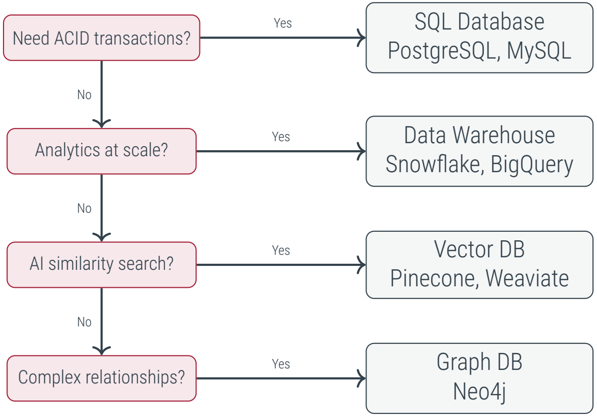
- Neo4j, Amazon Neptune
- Relationships, knowledge graphs
- Entity resolution, networks

Vector Databases:

- Pinecone, Weaviate, Qdrant
- Embeddings for RAG/AI
- Similarity search

Database Selection: Executive Decision Framework

Match the system to your constraints



Executive Rule of Thumb

Compute Infrastructure: CPU, GPU, and Beyond

Understanding what powers AI workloads

AI's compute demands are fundamentally different from traditional software.

CPU

- General-purpose
- Complex control flow
- Low parallelism (8-64 cores)
- Orchestration & logic

Best for: Traditional software, data processing, serving logic

GPU

- Massively parallel
- Matrix math optimized
- 10,000+ cores
- AI training & inference

Best for: Deep learning, large-scale number crunching

TPU/NPU

- AI-specialized silicon
- Even more efficient
- Google TPU, Apple NPU
- Vendor lock-in risk

Best for: Cloud AI services, edge inference

On "Quantum Computing":

- Not relevant for mainstream ML today
- Potential future niche: optimization, simulation
- Separate timeline from current AI ROI discussions

Cost Reality:
GPU compute is expensive.
H100: $25-50/hour (cloud)
Training GPT-4: $50-100M+

What Do We Mean by ”AI” Today?

AI as an umbrella term

AI is not one thing—it’s a spectrum of capabilities built on different techniques.

The AI Umbrella Includes:

- Rule-based automation — explicit logic
- Classical ML — statistical learning from data
- Deep learning — neural networks at scale
- Generative models — content creation
- Agentic systems — autonomous action

Executive Mental Model:

- Each layer builds on the previous
- More capability = more complexity
- Not all problems need the newest approach
- Match technique to problem

Key insight: ”AI” in your organization likely means multiple techniques coexisting.

The AI Capabilities Map

Three categories executives should remember

When evaluating AI initiatives, categorize by the type of capability being delivered.

Predictive

- Classification
- Forecasting
- Anomaly detection
- Risk scoring

"What will happen?"

Generative

- Text generation
- Code synthesis
- Image creation
- Document drafting

"Create something new"

Agentic

- Tool use
- Multi-step reasoning
- Autonomous workflows
- Decision execution

"Act under constraints"

Governance complexity increases left to right

What AI Is Not

Clearing misconceptions for better decisions

AI is NOT:

- Human reasoning — pattern matching, not understanding
- Guaranteed truth — probabilistic, can hallucinate
- Deterministic — same input can yield different outputs
- A strategy substitute — it's a capability, not a direction
- Set-and-forget — requires monitoring and maintenance

AI IS:

- Statistical pattern recognition at scale
- A tool that amplifies human capability
- Data-dependent — quality in, quality out
- An operational system requiring governance
- Rapidly evolving — capabilities change quarterly

Executive Principle

Why Now? The Convergence

Four forces enabling the current AI moment

AI has had multiple hype cycles. What's different this time?

1. Compute

- GPU acceleration
- Cloud scale
- 10,000× cheaper than 2012

2. Data

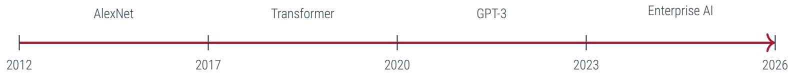
- Internet-scale text
- Digitized operations
- Labeled datasets

3. Algorithms

- Transformers (2017)
- Transfer learning
- Scaling laws

4. Distribution

- API access
- IDE integration
- Consumer adoption



AI History: Era A — Foundations (1940s–1960s)

The birth of artificial intelligence

The conceptual foundations were laid before computers were widely available.

Key Milestones:

- 1943: McCulloch & Pitts — first mathematical model of a neuron McCulloch and Pitts 1943
- 1950: Turing — "Computing Machinery and Intelligence" Turing 1950
- 1956: Dartmouth Workshop — term "Artificial Intelligence" coined McCarthy et al. 2006
- 1957–58: Rosenblatt — Perceptron Rosenblatt 1957

The Mood:

- Unbounded optimism
- "Machines will think within 20 years"
- Heavy government funding
- Symbolic AI dominates

Lesson: Initial timelines were wildly optimistic.

1943
1950
1956
1958

Era A: Foundations

AI History: Era B — Symbolic AI & Early Limits (1960s–1970s)

Logic-based reasoning and its boundaries

The dominant approach tried to encode human knowledge as rules and logic.

Symbolic AI Approach:

- Expert systems with hand-coded rules
- Logic-based reasoning
- Knowledge representation
- Natural language via grammar rules

1969 — Minsky & Papert Minsky and Papert 1969:

- Published critique of Perceptrons
- Showed fundamental limitations
- Neural network funding collapsed

Why It Hit Limits:

- Brittleness — rules couldn't handle edge cases
- Combinatorial explosion — complexity grew exponentially
- Knowledge acquisition bottleneck — experts couldn't articulate all rules
- No learning — systems couldn't improve from data

Executive Lesson

Rule-based systems work for narrow, well-defined domains. They fail when reality is messy, in-

AI History: Era C — AI Winters (1970s–1990s)

Boom, bust, and the expert systems era

Unmet promises led to funding collapses—twice.

First AI Winter (1974–1980):

- DARPA cut funding after failed promises
- ”AI can't deliver” sentiment
- Research continued quietly

Expert Systems Boom (1980s):

- Commercial success initially
- XCON saved DEC $40M/year
- Massive corporate investment

Second AI Winter (1987–1993):

- Expert systems proved expensive to maintain
- Rules became outdated quickly
- Couldn't adapt to changing business
- $1B+ in failed projects

Pattern Recognition:

- Hype → Investment → Unmet expectations → Collapse
- Sound familiar?

Survivor insight: The ideas weren't wrong—the compute, data, and algorithms weren't ready.

AI History: Era D – Statistical ML Era (1990s–2000s)

Data-driven learning takes over

INTELLIGENT FUTURES AI CONSULTANCY GLOBAL SOLUTIONS

The shift from ”programming knowledge” to ”learning from data” transformed the field.

Key Methods That Emerged:

- Support Vector Machines Cortes and Vapnik 1995
- Random Forests Breiman 2001
- Boosting Freund and Schapire 1997
- Bayesian methods – principled uncertainty

Why It Worked:

- Strong mathematical foundations
- Provable guarantees
- Interpretable (relatively)

The ”Data-Driven” Paradigm Shift:

- Don’t encode rules—learn patterns
- More data ▯ better models
- Features still hand-engineered
- Practical: spam filters, fraud detection, recommendations

Still Relevant Today:

- Many enterprise problems are best solved with these methods
- Interpretability matters for compliance

Executive note: These techniques remain the right choice for many tabular/structured data problems.

AI History: Era E — Deep Learning Revival (2006–2015)

Neural networks return with compute and data

The ideas from the 1980s finally had the infrastructure to work.

Key Breakthroughs:

- 2006: Hinton — Deep Belief Networks Hinton et al. 2006
- 2012: AlexNet Krizhevsky et al. 2012 — CNNs + GPUs crush competition
- 2014: GANs Goodfellow et al. 2014
- 2015: ResNet He et al. 2016 — very deep networks

What Changed:

- GPUs — 50× faster training
- Big data — ImageNet (14M labeled images)
- Better techniques — dropout, batch norm, ReLU
- Open source — reproducibility

The AlexNet Moment (2012)

AlexNet reduced ImageNet error rate from 26% to 15%—a discontinuous leap.

Lesson: When architecture + data + compute align, progress can be sudden and dramatic.

AI History: Era F — Transformers & Modern NLP (2017–2020)

Attention is all you need

A new architecture unlocked language understanding at scale.

2017: The Transformer Paper Vaswani et al. 2017

- "Attention Is All You Need" (Google)
- Replaced sequential processing with parallel attention
- Enabled much larger models
- Better at capturing long-range dependencies

What Followed:

- 2018: BERT Devlin et al. 2019
- 2019: GPT-2 Radford et al. 2019
- 2020: GPT-3 Brown et al. 2020

New Paradigm — Transfer Learning:

- Pretrain on massive text corpora
- Fine-tune for specific tasks
- Don't start from scratch

Executive Implications:

- Language tasks suddenly tractable
- Foundation models as starting point
- Context windows define capability limits
- Cost scales with model size

This is the architecture powering today's LLMs.

AI History: Era G — Foundation Models & Enterprise AI (2020–2023)

From research to products

AI moved from labs to widespread enterprise and consumer deployment.

Technical Advances:

- Instruction tuning Ouyang et al. 2022
- RLHF — alignment with human preferences
- Tool use — models can call APIs, search, calculate
- Multimodal — text + images + code

Key Releases:

- ChatGPT (Nov 2022) — 100M users in 2 months
- GPT-4 OpenAI 2023 — multimodal reasoning
- GitHub Copilot — AI in developer workflows

Enterprise Reality:

- Governance becomes central — who controls what AI does?
- Data privacy concerns — where does my data go?
- Integration challenges — connecting to existing systems
- ROI questions — beyond demos to measurable value

Gap Emerges:

- Demo-to-production is hard
- Hallucination is a real problem
- Evaluation is immature

AI History: Era H — Efficiency & Systems (2023–2026)

Where we are now

The frontier has shifted from "bigger models" to "better systems."

Efficiency Breakthroughs:

- Mixture-of-Experts DeepSeek-AI 2024; Jiang et al. 2024
- Distillation — smaller models learn from larger ones
- Quantization — reduce precision, maintain quality
- Open models — Llama Touvron et al. 2023, Mistral, DeepSeek

Cost Control Matters:

- Token costs dropped 100× in 2 years
- Small models often sufficient
- Self-hosting becomes viable

Systems Engineering Dominates:

- RAG Lewis et al. 2020 — retrieval-augmented generation
- Evaluation discipline — measure before deploy
- Workflow integration — AI in processes, not standalone
- Agentic patterns — bounded autonomy with guardrails

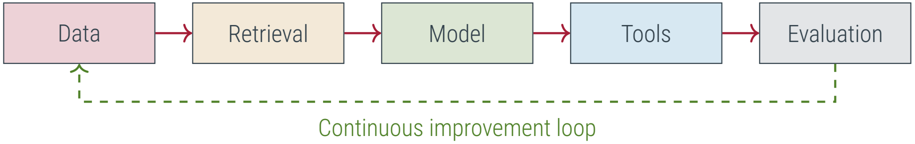
Executive Implication:

- Model selection is a cost/capability trade-off
- Architecture > model size
- Evaluation is competitive advantage

Bridge: AI Is a Systems Discipline Now

Setting up the deep dive

A working AI product is much more than a model.



The Full Stack:

- Data pipelines & quality
- Retrieval & knowledge systems
- Model selection & prompting
- Tool integration & guardrails
- Evaluation & monitoring

What This Means for You:

- Don’t just ”buy a model”
- Invest in data & evaluation
- Architect for iteration
- Govern the whole system

Next: We’ll go deep on how these components actually work.

The Three Paradigms of Machine Learning

How machines learn from data

All ML methods fall into three fundamental learning paradigms—each suited to different business problems.

Supervised

Learning from labeled examples

- Input ▯ Known output
- Learn the mapping
- Predict on new data

Classification, regression, forecasting

Unsupervised

Finding structure in data

- No labels provided
- Discover patterns
- Group similar items

Clustering, anomaly detection, compression

Reinfortic Learning

Learning from rewards

- Sequential decisions
- Trial and error
- Maximize long-term reward

Games, robotics, recommendations

Executive insight: 90%+ of enterprise ML is supervised learning on structured data.

Supervised Learning: The Workhorse of Enterprise AI

Classification and regression

Most business AI problems are supervised: you have historical data with known outcomes.

Classification — Discrete categories

- Will this customer churn? (Yes/No)
- Is this transaction fraud? (Yes/No)
- What topic is this email? (Sales/Support/Spam)
- Which product to recommend? (A/B/C/...)

Output: probabilities across categories

Regression — Continuous values

- What will revenue be next quarter?
- How long until this machine fails?
- What price maximizes profit?
- How many units will we sell?

Output: a number (with uncertainty)

The Supervised Learning Recipe

1. Collect historical data with known outcomes (labels)
2. Train model to find patterns connecting inputs to outputs
3. Validate on held-out data to estimate real-world performance
4. Deploy and monitor for drift

Unsupervised & Reinforcement Learning

When labels are unavailable or actions matter

Unsupervised Learning

No labels—find structure in data itself

Key techniques:

- Clustering – group similar customers, documents, behaviors
- Dimensionality reduction – compress features, visualize high-dim data (PCA, t-SNE)
- Anomaly detection – find outliers without labeled fraud cases

Use when: You don't have labels, or want to discover unknown patterns

Reinforcement Learning (RL)

Learn optimal actions through trial and error

Key characteristics:

- Sequential decisions – actions affect future states
- Delayed rewards – outcome known only later
- Exploration vs exploitation – try new vs use known

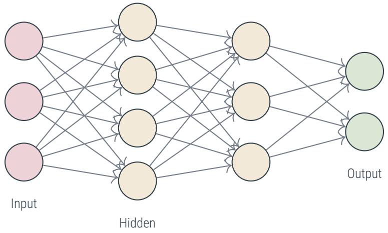
Use when: Optimizing multi-step processes (pricing, routing, control)

Note: RL is powerful but harder to productize. Start with supervised if you have labels.

Multi-Layer Perceptrons (MLPs): The Foundation

The simplest neural network architecture

An MLP is a stack of layers that learn to transform inputs into outputs through nonlinear functions.



How it works:

- Each connection has a learnable weight
- Each layer applies: output $= \sigma(Wx + b)$
- σ is a nonlinearity (ReLU, sigmoid)
- Training: adjust weights to minimize prediction error

Key insight:

- Can approximate any function (universal approximation)
- More layers = more expressive power
- But: needs lots of data to avoid overfitting

MLPs in Practice: When to Use Them

Strengths, weaknesses, and enterprise applications

Strengths:

- Flexible function approximation
- Works on tabular data (structured)
- Easy to implement and train
- Foundation for all deep learning

Weaknesses:

- Often outperformed by tree-based methods on tabular data (XGBoost, Random Forest)
- No built-in structure for images, text, sequences
- Can overfit without regularization

Enterprise Use Cases:

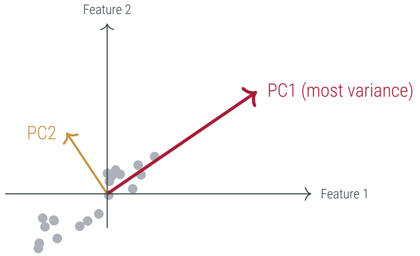
- Churn prediction – customer features ▯ churn probability
- Credit scoring – financial data ▯ risk score
- Demand forecasting – historical features ▯ units sold
- Fraud detection – transaction features ▯ fraud probability

Executive rule:
For tabular data, try gradient-boosted trees (XGBoost) first—often better with less tuning.

PCA: Dimensionality Reduction

Compressing data while preserving information

Principal Component Analysis (PCA) finds the directions of maximum variance in your data.



Original: 2 dimensions

PCA finds directions of spread

What PCA Does:

- Finds orthogonal axes (principal components)
- Ranked by variance explained
- Project data onto top k components
- Lossy compression: keep signal, reduce noise

Use Cases:

- Reduce 1000 features to 50 for faster training
- Visualize high-dimensional data in 2D/3D
- Remove noise from sensor data
- Feature engineering before ML

Limitation: PCA only captures linear relationships.

PCA Example: Customer Behavior Analysis

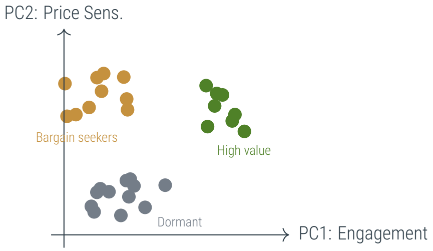
From 50 metrics to actionable segments

The Scenario:

- Marketing has 50 customer metrics
- Purchase frequency, recency, categories, channel preferences, engagement scores...
- Too many dimensions to visualize or interpret

PCA Reveals:

- PC1 (40% variance): "Overall engagement"
- PC2 (15% variance): "Price sensitivity"
- PC3 (10% variance): "Channel preference"
- First 3 components capture 65% of information



Result: Clear segments emerge from compressed representation

Caveat: Components are interpretable only if you examine the loadings (which original features contribute).

t-SNE: Visualizing Complex Data

Nonlinear dimensionality reduction for exploration

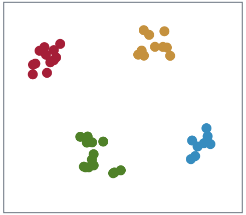
t-SNE (t-distributed Stochastic Neighbor Embedding) preserves local neighborhoods when projecting to 2D.

How t-SNE Differs from PCA:

- PCA: Linear projection, preserves global variance
- t-SNE: Nonlinear, preserves local similarity
- Points that are similar stay close
- Reveals clusters that PCA might miss

Use Cases:

- Visualizing embeddings (words, documents, images)
- Exploring customer segments
- Quality check on clustering results


t-SNE projection of 100-dim data

Clusters clearly separated

t-SNE: Critical Warnings for Executives

What t-SNE cannot tell you

van der Maaten & Hinton, 2008

t-SNE is NOT:

- Distance-preserving – distances between clusters are meaningless
- Deterministic – different runs give different layouts
- A clustering algorithm – it only visualizes, doesn't assign labels
- Suitable for quantitative analysis – don't measure cluster sizes/gaps

t-SNE IS:

- Great for exploration and hypothesis generation
- Useful to sanity-check other analyses
- A way to communicate structure visually

Executive Rule

Use t-SNE for qualitative exploration only.

Never make business decisions based on:

- Cluster sizes in t-SNE plots
- Distances between clusters
- Apparent "gaps" in the data

Always validate with quantitative methods.

Modern alternative: UMAP—faster and better preserves global structure, but same caveats apply.

Convolutional Neural Networks (CNNs)

The architecture that conquered computer vision

INTELLIGENT FUTURES AI CONSULTANCY GLOBAL SOLUTIONS

van der Maaten & Hinton, 2008

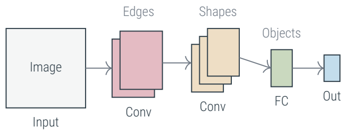
CNNs revolutionized image processing by learning spatial hierarchies of features.

Key Innovation:

- Local receptive fields – each neuron sees only a small patch
- Weight sharing – same filter applied everywhere
- Hierarchical features – edges □ shapes □ objects
- Far fewer parameters than fully connected

The AlexNet Moment (2012) Krizhevsky et al. 2012:

- ImageNet error: 26% □ 15%
- Discontinuous improvement
- CNN + GPU + Big Data = breakthrough



Insight: CNN layers learn increasingly abstract features automatically.

CNNs in Enterprise: Where They Still Dominate

Practical applications beyond research

van der Maaten & Hinton, 2008

Manufacturing & Quality:

- Defect detection – visual inspection at scale
- Quality control – surface anomalies, assembly verification
- Predictive maintenance – analyze equipment images

Document Processing:

- OCR – text extraction from images
- Document classification – invoices, receipts, forms
- Signature verification

Healthcare & Medical:

- Medical imaging – X-rays, MRIs, pathology slides
- Diagnostic assistance – detect anomalies, measure features

Retail & Security:

- Visual search – find similar products
- Inventory tracking – shelf monitoring
- Access control – facial recognition

Executive Lesson from CNNs

The AlexNet breakthrough taught us: when architecture + data + compute align, progress can be

What Neural Networks Actually Learn

Representations are the key insight

van der Maaten & Hinton, 2008

The magic of deep learning: networks learn to represent data, not just classify it.

Traditional ML:

- Humans engineer features
- ”Age, income, purchase count...”
- Model learns weights on fixed features
- Quality depends on feature design

Feature engineering is manual and domain-specific

Deep Learning:

- Network learns features automatically
- Hidden layers = learned representations
- ”Embedding” = useful compressed form
- Transfers across tasks

Representation learning scales with data

Key Insight

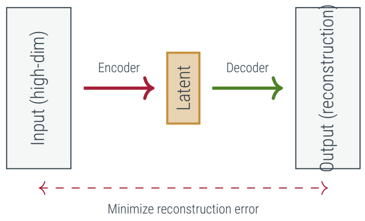
The representation layer (embeddings) is often more valuable than the final output.
A good representation can be reused for many downstream tasks.

Autoencoders: Learning to Compress

The encoder-decoder architecture

van der Maaten & Hinton, 2008

An autoencoder learns to compress data into a small representation, then reconstruct it.



How It Works:

- Encoder: Compress input to small ”latent” vector
- Bottleneck: Forces network to learn essential features
- Decoder: Reconstruct original from latent
- Training: Minimize reconstruction error

The Insight:

- If it can reconstruct, latent must capture meaning
- Latent = compressed representation

Autoencoder Applications

Compression, denoising, and anomaly detection

van der Maaten & Hinton, 2008

Compression

- Reduce data dimensionality
- Store latent vectors instead of raw data
- Faster downstream processing

Example: Compress 1000 features to 50

Denoising

- Train on noisy □ clean pairs
- Network learns to remove noise
- Extracts underlying signal

Example: Clean sensor data, audio

Anomaly Detection

- Train only on ”normal” data
- Anomalies = high reconstruction error
- No labeled anomalies needed!

Example: Fraud, equipment failure

Anomaly Detection Pattern

1. Train autoencoder on normal operations only
2. In production: if reconstruction error > threshold □ flag as anomaly
3. Key advantage: Works without labeled fraud/failure cases

Autoencoder Example: Industrial Anomaly Detection

Detecting equipment failures without labeled failure data

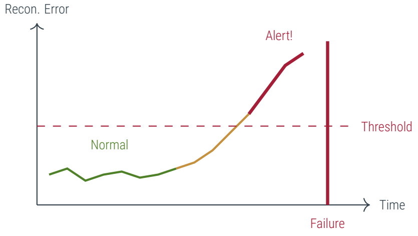
van der Maaten & Hinton, 2008

The Scenario:

- Manufacturing equipment with 100 sensors
- Failures are rare (good!)
- But: no labeled failure data to train on
- Need: early warning system

Autoencoder Solution:

- Train on months of normal operation
- Network learns "what normal looks like"
- Pre-failure: sensors drift from normal
- Autoencoder can't reconstruct abnormal patterns
- High error = early warning



Result: Days of early warning before failure, enabling preventive maintenance.

Bridge: This encoder→latent→decoder pattern is fundamental to modern generative AI.

From Autoencoders to Generative AI

The conceptual bridge to LLMs and diffusion models

van der Maaten & Hinton, 2008

Modern generative models build on the encoder-decoder concept.

Autoencoder Paradigm:

- Encoder: Input ▯ compressed representation
- Decoder: Representation ▯ reconstruct input
- Goal: faithful reconstruction

Generative Insight:

- What if we only use the decoder?
- Feed it a representation ▯ generate output
- Don't reconstruct—create something new

Modern Architectures:

- VAEs: Learn a structured latent space, sample to generate
- Transformers (decoder-only): Generate text token by token Vaswani et al. 2017
- Diffusion models: Learn to denoise, generate by iterative denoising

All share the concept: learned representations enable generation

Conceptual Link

Encoder produces embeddings (representations)

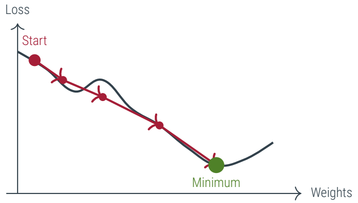
Decoder generates outputs conditioned on representations

Training Neural Networks: The Optimization Challenge

How models learn from data

van der Maaten & Hinton, 2008

Training = finding weights that minimize prediction error on training data.



Gradient descent: follow the slope downhill

Gradient Descent Rumelhart et al. 1986:

- Compute error (loss) on batch of data
- Calculate gradient: which direction reduces loss?
- Update weights: small step in that direction
- Repeat millions of times

Key Hyperparameters:

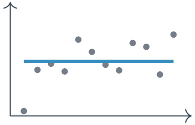
- Learning rate: Step size (too big = overshoot, too small = slow)
- Batch size: Samples per gradient update
- Epochs: Passes through full dataset

Overfitting vs Underfitting

The fundamental tradeoff in machine learning

van der Maaten & Hinton, 2008

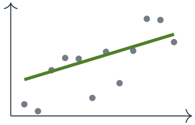
Underfitting



- Model too simple
- Misses patterns in data
- High error on both train & test

Fix: more capacity, features, training

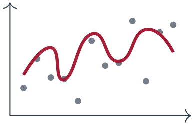
Good Fit



- Captures true pattern
- Ignores noise
- Generalizes to new data

Goal: this is what we want

Overfitting



- Model memorizes training data
- Fits noise, not signal
- Fails on new data

The silent killer of ML projects

Executive insight: A model that looks perfect on training data may be worthless in production. Always evaluate on held-out test data.

ML Failure Modes Executives Must Know

What causes AI projects to fail in production

van der Maaten & Hinton, 2008

Data Leakage – The Silent Killer

- Information from future/test leaks into training
- Model appears perfect but fails in production
- Examples:
  - Using outcome data as input feature
  - Time-series split done wrong
  - Same customer in train and test

Distribution Shift

- Production data differs from training data
- Model degrades over time
- Causes: Seasonality, market changes, new user segments

Label Quality Issues

- Garbage in = garbage out
- Inconsistent labeling
- Missing or delayed labels

Evaluation Leakage

- Test set used repeatedly for tuning
- Overfitting to evaluation benchmark
- ”Teaching to the test”

Governance Requirement

Embeddings: The Foundation of Modern AI

Mapping discrete objects to meaningful vectors

van der Maaten & Hinton, 2008

An embedding maps discrete items (words, products, users) to vectors where geometry encodes meaning.

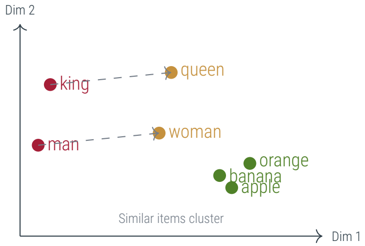
The Core Idea:

- Words/items ▯ vectors of numbers
- Similar items ▯ nearby vectors
- Relationships preserved geometrically
- Enables math on concepts

Classic Example Mikolov et al. 2013:

$$\vec{\text{king}} - \vec{\text{man}} + \vec{\text{woman}} \approx \vec{\text{queen}}$$

Vector arithmetic captures semantic relationships



What Can Be Embedded?

Embeddings work for almost any discrete data

van der Maaten & Hinton, 2008

Text Embeddings:

- Words, sentences, documents
- Enable semantic search
- Power RAG systems
- Compare meaning, not keywords

Image Embeddings:

- Images ▯ vectors via CNN/ViT
- Visual similarity search
- Reverse image search
- Content moderation

User/Product Embeddings:

- Collaborative filtering
- "Users like you bought..."
- Personalized recommendations

Code Embeddings:

- Functions ▯ vectors
- Find similar code
- Semantic code search
- Duplicate detection

Key insight: Embeddings are the universal interface. Text, images, users, products—all become vectors that can be compared, clustered, and retrieved.

Embeddings Example: Semantic Search

Finding relevant documents by meaning, not keywords

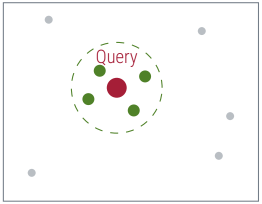
van der Maaten & Hinton, 2008

Traditional Keyword Search:

- Query: ”vacation policy”
- Matches: documents containing ”vacation” AND ”policy”
- Misses: ”PTO guidelines”, ”time off procedures”, ”leave entitlement”

Semantic Search with Embeddings:

- Query embedding vector
- Find documents with similar vectors
- Finds: All semantically related docs, regardless of exact wording



Embedding space

Retrieve k nearest neighbors

Enterprise Value

Embedding-based search is the backbone of RAG systems.

Transformers: The Architecture Behind LLMs

Why "Attention Is All You Need"

van der Maaten & Hinton, 2008

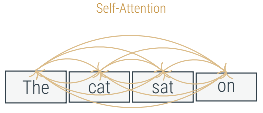
The Transformer architecture Vaswani et al. 2017 revolutionized NLP and enabled today's large language models.

Before Transformers (RNNs/LSTMs):

- Process text sequentially, word by word
- Hard to parallelize slow training
- Long-range dependencies get "forgotten"
- Limited context window

The Transformer Innovation:

- Process all tokens in parallel
- Attention: Each token can "look at" all others
- No sequential bottleneck
- Scales to massive models



Every token attends to every token

Result: Training that took weeks now takes hours. Models can be 1000× larger.

The Attention Mechanism

Content-addressable retrieval within the input

van der Maaten & Hinton, 2008

Attention lets the model dynamically decide which parts of the input are relevant to each output.

Intuition:

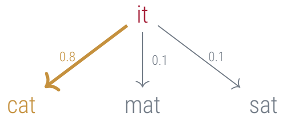
- For each token, ask: "What should I pay attention to?"
- Compute relevance scores to all other tokens
- Weight information by relevance
- Aggregate: weighted sum of values

The Q-K-V Framework:

- Query (Q): "What am I looking for?"
- Key (K): "What do I contain?"
- Value (V): "What information do I provide?"
- Score = Query · Key (dot product)

Example – Resolving "it":

"The cat sat on the mat because it was tired."



Attention learns that "it" refers to "cat" with high probability

Transformer Variants: Encoder, Decoder, and Both

Different architectures for different tasks

van der Maaten & Hinton, 2008

Encoder-Only

(BERT-style) Devlin et al. 2019

- Bidirectional attention
- Sees full input at once
- Best for: understanding

Tasks: Classification, NER, embeddings, similarity

Decoder-Only

(GPT-style) Brown et al. 2020

- Causal attention (left-to-right)
- Generates token by token
- Best for: generation

Tasks: Text generation, chat, code, reasoning

Encoder-Decoder

(T5-style)

- Encoder reads input
- Decoder generates output
- Best for: transformation

Tasks: Translation, summarization, Q&A

What You're Using Today

ChatGPT, Claude, Gemini, Llama = Decoder-only transformers
They generate text left-to-right, predicting the next token given all previous tokens.

How LLMs Generate Text

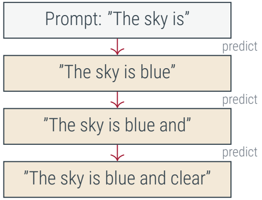
From prompt to completion

van der Maaten & Hinton, 2008

Text generation is iterative: predict next token, append, repeat.

The Generation Loop:

1. Encode prompt into token IDs
2. Forward pass: compute probability distribution over vocabulary
3. Sample next token (with temperature)
4. Append token to sequence
5. Repeat until stop token or max length



Temperature Controls Randomness:

- $T \to 0$: Always pick most likely (deterministic)
- $T = 1$: Sample from learned distribution
- $T > 1$: More random/creative

Key insight: LLMs don't "understand"—they predict statistically likely continuations based on training data patterns.

Context Windows: The Memory Limit

What the model can "see" at once

van der Maaten & Hinton, 2008

The context window is the maximum number of tokens (prompt + response) the model can process.

Context Window Evolution:

- GPT-3 (2020): 4K tokens
- GPT-4 (2023): 8K–128K tokens
- Claude 3 (2024): 200K tokens
- Gemini 1.5 (2024): 1M+ tokens

What's a Token?

- Roughly 0.75 words in English
- 4K tokens ≈ 3,000 words ≈ 6 pages
- 128K tokens ≈ a short book

Why It Matters:

- Larger context = more information available
- Can include more documents, longer conversations
- But: Compute and cost scale with context

The Trade-offs:

- Cost: Proportional to tokens processed
- Latency: Longer context = slower response
- Quality: "Lost in the middle" problem

Context Window Strategy: Less Is Often More

Smart retrieval beats context stuffing

van der Maaten & Hinton, 2008

Naive Approach:

"Dump everything into context"

- Include all potentially relevant docs
- Max out the context window
- Let the model figure it out

Problems:

- High cost (pay per token)
- Slower responses
- Model gets distracted by irrelevant info
- "Lost in the middle" — info in middle gets ignored

Smart Approach:

"Retrieve only what's relevant"

- Use embeddings to find relevant passages
- Include only top-k most relevant
- Keep context focused and concise

Benefits:

- Lower cost
- Faster responses
- Better answer quality
- Clearer citation/attribution

Design Principle

[illegible]

Tool Use: Extending LLM Capabilities

When generation isn't enough

van der Maaten & Hinton, 2008

INTELLIGENT FUTURES
AI CONSULTANCY
GLOBAL SOLUTIONS

LLMs can learn to call external tools—calculators, APIs, databases—to overcome their limitations.

Why Tool Use?

- LLMs are bad at math □ call calculator
- LLMs have stale knowledge □ call search API
- LLMs can't access your data □ call database
- LLMs can't take actions □ call business APIs

How It Works:

- Model outputs structured tool call
- System executes tool, returns result
- Model continues with result in context

Example Flow:

1. User: "What's our Q3 revenue?"
2. Model decides: `query_database("Q3 revenue")`
3. System executes query □ returns "$4.2M"
4. Model responds: "Your Q3 revenue was $4.2M"

Common Tools:

- Code execution (Python interpreter)
- Web search
- Database queries
- API calls (CRM, ERP, etc.)

Agentic Patterns: Bounded Autonomy

Multi-step reasoning with guardrails

van der Maaten & Hinton, 2008

Agents combine LLMs + tools + planning to accomplish complex tasks autonomously.

What Makes an Agent:

- Planning: Break task into steps
- Tool use: Execute actions
- Memory: Track progress and context
- Reflection: Evaluate and adjust

Example – Research Agent:

1. Plan: "Need 3 competitor analyses"
2. Search: Query web for each competitor
3. Analyze: Extract key info
4. Synthesize: Compile report
5. Reflect: "Is this complete?"

Governance Requirements:

- Permissions: What can the agent access?
- Audit: Log all tool calls and decisions
- Limits: Max steps, cost caps, time bounds
- Human-in-loop: Approval for sensitive actions
- Fail-safes: What if agent goes off-track?

Executive Caution

Part II-D: RAG Systems in Detail

25 minutes — The primary lever for enterprise value

van der Maaten & Hinton, 2008

Retrieval-Augmented Generation

The architecture pattern that makes LLMs useful for enterprise knowledge

What We'll Cover:

1. Why RAG exists — the parametric memory problem
2. The canonical RAG pipeline — 9 stages
3. RAG variants — from naive to enterprise-grade
4. Evaluation — measuring what matters
5. Example walkthrough — a realistic enterprise query

Why This Matters

RAG is how enterprises get accurate, grounded, auditable answers
from LLMs about their own data. Get this right ▯ unlock value. Get it wrong ▯ liability

Why RAG Exists: The Parametric Memory Problem

Models are not databases

van der Maaten & Hinton, 2008

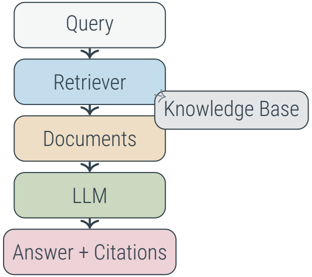
The Problem with ”Parametric Memory”:

- LLMs store knowledge in weights
- Training data has a cutoff date
- Can’t reliably recall specific facts
- No access to your proprietary data
- Can’t cite authoritative sources

What Happens Without RAG:

- ”Who is our CFO?” □ Hallucination
- ”What’s our refund policy?” □ Outdated info
- ”Show me Q3 numbers” □ Made up

RAG = Retrieval + Generation + Citations

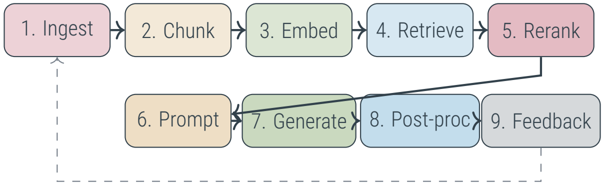


Key insight: Retrieve relevant context at query time, don’t rely on model’s memory.

The Canonical RAG Pipeline: Overview

Nine stages from document to answer

van der Maaten & Hinton, 2008



Offline (Indexing):

1. Ingest: Parse docs, extract metadata
2. Chunk: Split into retrievable units
3. Embed: Convert to vectors, index

Online (Query):

4. Retrieve: Find similar chunks
5. Rerank: Improve precision
6. Prompt: Assemble context
7. Generate: LLM produces answer
8. Post-process: Validate, format
9. Feedback: Log, evaluate, improve

RAG Pipeline: Ingestion & Chunking

The foundation that determines success or failure

van der Maaten & Hinton, 2008

1. Ingestion – What to capture:
   - Content: Text, tables, images
   - Metadata: Owner, date, source, version
   - Permissions: ACLs, classification level
   - Structure: Headers, sections, hierarchy

Common Failures:

- Tables rendered as gibberish
- PDFs with OCR errors
- Missing permission metadata
- Stale documents not removed

2. Chunking – Strategy matters:

Strategy	Best For
Fixed-size	Simple, predictable
Sentence-based	Natural boundaries
Paragraph-based	Coherent units
Section-based	Structured docs
Semantic	Topic coherence
Overlapping	Context preservation

Rule of thumb: Chunk size should match typical query scope. Too small □ missing context. Too large □ noise + cost.

RAG Pipeline: Embedding, Retrieval, Reranking

Finding the needle in the haystack

van der Maaten & Hinton, 2008

3. Embedding + Indexing:
   - Convert chunks to vectors
   - Store in vector index
   - Enable similarity search

Index Options:

- Dedicated vector DB
- Vector extension in existing DB
- Hybrid with keyword index

4. Retrieval:
   - Query ▯ embedding
   - Find top-k similar chunks
   - Apply filters (permissions, date, source)

Recall vs Precision:

- High k ▯ more recall, more noise
- Low k ▯ might miss relevant info
- Balance via reranking

5. Reranking:
   - Take top-N candidates
   - Score with cross-encoder
   - Return top-K highest

Why Rerank?

- Embeddings = fast but approximate
- Reranker = slower but precise
- Retrieve 50 ▯ Rerank to 5

Hybrid Retrieval

Combine vector search (semantic similarity) with keyword search (exact terms)

RAG Pipeline: Generation & Post-Processing

From context to trustworthy answer

van der Maaten & Hinton, 2008

6. Prompt Assembly:
   - System instructions (persona, constraints)
   - Retrieved context (with source markers)
   - User query
   - Output format specification
7. Generation:
   - LLM produces answer using context
   - Key instruction: "Only use provided context"
   - Citation markers: [Source 1], [Doc A]
   - Refusal when uncertain or no relevant context
8. Post-Processing:
   - Format validation: JSON schema, required fields
   - Citation verification: Do citations match sources?
   - PII scrubbing: Remove leaked sensitive data
   - Safety checks: Content policy compliance
9. Feedback Loop:
   - Log query, retrieval, response
   - Capture user feedback (thumbs up/down)
   - Build evaluation dataset
   - Identify retrieval failures

Executive insight: The feedback loop is how RAG systems improve over time. Without it, you're flying blind.

RAG Variant A: Naive RAG

Simple but limited

van der Maaten & Hinton, 2008

How It Works:

1. Embed query
2. Vector search □ top-k chunks
3. Stuff all chunks into prompt
4. Generate answer

When It’s Sufficient:

- Small, homogeneous document set
- Simple factual queries
- Low-stakes use cases
- Proof of concept / demos

Failure Modes:

- Irrelevant retrieval: Semantic similarity ≠ relevance
- Hallucination despite context: Model ignores or misinterprets
- Poor chunking: Context split across chunks
- No ranking: Garbage in first position
- No permissions: Returns unauthorized content

Executive Guidance

Naive RAG is a starting point, not a production architecture.

RAG Variant B: Hybrid Retrieval

Keyword + Vector = Best of both

van der Maaten & Hinton, 2008

The Problem:

- Vector search: great for semantic similarity
- But fails on: exact terms, IDs, codes, names
- ”Find policy ABC-123” � vector search returns wrong policy

The Solution:

- Run both keyword (BM25) and vector search
- Combine results with reciprocal rank fusion
- Rerank the combined set

When to Use Hybrid:

- Legal/compliance: Exact clause references
- Technical docs: Error codes, product IDs
- Financial: Account numbers, ticker symbols
- HR: Policy numbers, form names

Implementation:

- Elasticsearch + vector plugin
- PostgreSQL + pgvector + FTS
- Dedicated hybrid search services

Rule of thumb: If your corpus has important exact-match terms, hybrid is not optional.

RAG Variant C: Hierarchical RAG

Coarse-to-fine retrieval

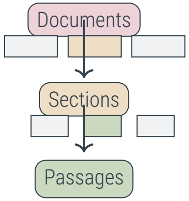
van der Maaten & Hinton, 2008

The Problem:

- Flat retrieval loses document structure
- Similar passages from different docs get mixed
- Hard to trace "which document said this"

Hierarchical Approach:

1. Level 1: Retrieve relevant documents
2. Level 2: Within docs, retrieve sections
3. Level 3: Within sections, retrieve passages



Benefits:

- Better traceability
- Reduces context noise
- Respects document boundaries

Best for: Large document collections with clear structure (manuals, policies, legal).

RAG Variant D: Multi-Query RAG

Multiple perspectives, better recall

van der Maaten & Hinton, 2008

The Problem:

- User query may be ambiguous
- Single embedding may miss relevant docs
- Different phrasings match different content

Multi-Query Approach:

1. LLM generates 3-5 query variations
2. Run retrieval for each variation
3. Merge and deduplicate results
4. Rerank combined set

Example:

Original: ”How do I get reimbursed?”

Generated variations:

- ”expense reimbursement process”
- ”submit expenses for payment”
- ”travel expense policy”
- ”reimbursement form submission”

Trade-offs:

- Higher latency (multiple retrievals)
- Higher cost (LLM for query gen)
- Risk of query drift

Best for: Ambiguous queries, diverse document language, high-stakes answers where recall matters.

RAG Variant E: GraphRAG / Knowledge Graph

Beyond embedding similarity

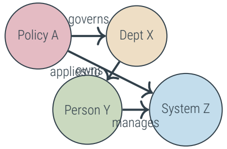
van der Maaten & Hinton, 2008

The Problem:

- Embeddings capture similarity, not relationships
- ”Who reports to whom?” needs structure
- Multi-hop reasoning across entities

GraphRAG Approach:

1. Extract entities from documents
2. Build relationships (owns, reports-to, depends-on)
3. Query combines graph traversal + vector search
4. Context includes entity relationships



Strong For:

- Organizational knowledge
- Ownership and accountability
- Dependency tracking
- Compliance traceability

Investment required: Entity extraction, schema design, ongoing maintenance. High value but high cost.

RAG Variant F: Text-to-SQL / Structured RAG

When the answer lives in a database

van der Maaten & Hinton, 2008

The Problem:

- Many enterprise answers are in databases
- "What was Q3 revenue?" needs SQL, not document retrieval
- RAG over documents can't give precise KPIs

Text-to-SQL Approach:

1. Natural language query
2. LLM generates SQL (with schema context)
3. Execute SQL against database
4. LLM explains results

Example Flow:

User: "Top 5 customers by revenue this quarter"

Generated SQL:

```
SELECT customer, SUM(revenue)
FROM orders
WHERE quarter = 'Q3'
GROUP BY customer
ORDER BY 2 DESC LIMIT 5;
```

Critical:

- SQL validation before execution
- Permission checks
- Query cost/timeout limits

Best for: Analytics, KPIs, operational metrics where correctness matters and data is structured.

RAG Variant G: Code/Repository RAG

Specialized retrieval for software artifacts

van der Maaten & Hinton, 2008

The Challenge:

- Code has different structure than prose
- Functions, classes, imports, call graphs
- Need to retrieve relevant code context

What to Index:

- Source code (functions, classes)
- Documentation (docstrings, README)
- Architecture Decision Records (ADRs)
- Issue tickets and PRs
- API specifications

Chunking Strategies for Code:

- Function-level: Natural code units
- Class-level: Object context
- File-level: Module context
- Dependency-aware: Include imports
- Call-graph aware: Related functions

Use Cases:

- "How does authentication work?"
- "Find usages of deprecated API"
- "What does this error mean?"

Key insight: Code assistants are RAG systems with specialized indexing and retrieval for software.

Enterprise RAG: Permissions & Audit

The governance layer that makes RAG safe

van der Maaten & Hinton, 2008

The Risk:

- RAG can expose unauthorized data
- "Summarize all HR docs" � returns confidential info
- UI-level permissions are not enough

Permission Enforcement:

- At indexing: Store ACLs with chunks
- At retrieval: Filter by user permissions
- At generation: Don't mix authorization levels

Audit Requirements:

- What was retrieved? Source IDs, chunks
- What was generated? Full response
- Who asked? User identity
- When? Timestamp
- Why these sources? Relevance scores

Enables:

- Compliance investigation
- Quality debugging
- Continuous improvement

Executive Mandate

Every RAG deployment must answer: "What sources influenced this answer?"

Evaluating RAG Systems

Measuring what matters

van der Maaten & Hinton, 2008

Retrieval Metrics:

- Recall@k: Did we retrieve the right docs?
- Precision@k: How much noise in top-k?
- MRR: Where does first relevant doc appear?
- nDCG: Ranking quality

Key insight: If retrieval fails, generation can't succeed. Measure retrieval separately.

Building Your Evaluation Set:

1. Collect realistic queries from users
2. Identify authoritative source for each answer
3. Create "golden answers" with expert review
4. Track which sources should be retrieved
5. Run automated + human evaluation regularly

End-to-End Metrics:

- Groundedness: Is answer supported by sources?
- Citation accuracy: Do citations match claims?
- Factuality: Is answer correct?
- Completeness: Did we answer the question?
- Refusal behavior: Does it refuse appropriately?

RAG Example: Enterprise Policy Query

A realistic walkthrough

van der Maaten & Hinton, 2008

User Query: "What is our policy for remote work, and who approves exceptions?"

1. Retrieval Results:
   - ✓ HR-Policy-2024-Remote-Work.pdf (score: 0.92)
   - ✓ Exception-Approval-Process.docx (score: 0.87)
   - × Travel-Policy.pdf (score: 0.71, filtered)
2. Reranking:
   - ▶ Cross-encoder promotes Exception doc
   - ▶ Final context: 2 docs, 4 chunks
3. Generated Answer:
   "Our remote work policy (HR-2024-RW) allows up to 3 days/week remote for eligible roles. [Source 1, §2.1]
   Exceptions require approval from your department head and HR Business Partner. [Source 2, §4.2]
   Submit exception requests via ServiceNow."
4. Audit Log: Query ID, User, Timestamp, Sources used, Confidence scores

Key features: Citations, specific references, actionable answer, traceable to sources.

RAG: Executive Summary

Key takeaways for decision-makers

van der Maaten & Hinton, 2008

RAG Is The Enterprise Pattern:

- Grounds LLMs in your knowledge
- Enables citations and audit
- Keeps data current without retraining
- Respects permissions and compliance

Choose Your Variant:

- Start with hybrid retrieval
- Add reranking for precision
- Use hierarchical for large doc sets
- Add GraphRAG for relationships
- Use Text-to-SQL for structured data

Non-Negotiable Requirements:

- Permission enforcement at retrieval
- Complete audit logging
- Evaluation set and metrics
- Feedback loop for improvement

Success Criteria:

- Can you trace every answer to sources?
- Can you measure retrieval quality?
- Can you prove permission compliance?
- Can you show improvement over time?

The Bottom Line

Evaluation Discipline

How to know if your AI system actually works

What We'll Cover:

1. Why benchmarks matter — and their limits
2. Train/validation/test splits — the foundation of trust
3. Production monitoring — because deployment is just the beginning

Why This Matters

Without rigorous evaluation, you can't distinguish a working system from a lucky demo. Evaluation governance is as important as model selection.

Why Benchmarks Matter

The common language of AI capabilities

van der Maaten & Hinton, 2008

INTELLIGENT FUTURES
AI CONSULTANCY
GLOBAL SOLUTIONS

Benchmarks Drive Decisions:

- Vendor selection: "Model X scores 90% on MMLU"
- Progress tracking: "We improved 5% on our task"
- Research direction: Community focuses on benchmark gaps
- Investment: Benchmark gains attract funding

Common Benchmarks:

- MMLU: Multi-task language understanding
- HumanEval: Code generation
- MATH: Mathematical reasoning
- TruthfulQA: Factual accuracy

Critical Caveat:

Benchmark performance $\neq$
Your business task performance

Why The Gap Exists:

- Your data distribution differs
- Your success criteria differ
- Your failure costs differ
- Benchmark contamination in training

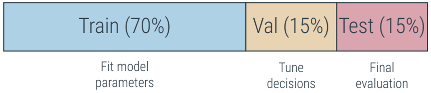
Executive implication: Use benchmarks for initial screening, but build your own evaluation set for

Train / Validation / Test Splits

The foundation of trustworthy evaluation

van der Maaten & Hinton, 2008

The Three-Way Split:



Purpose of Each:

- Training: Model learns from this data
- Validation: Guide hyperparameter choices, early stopping
- Test: Final, unbiased performance estimate

Leakage — The Silent Killer:

- Temporal: Future data in training
- Identity: Same customer in train/test
- Duplicate: Same example appears twice
- Feature: Target encoded in features

Symptoms of Leakage:

- "Too good to be true" test scores
- Model fails in production
- Performance degrades over time

Rule: Test set should never influence any decision during development. Touch it once, at the end.

Evaluation Governance

Process discipline for trustworthy results

van der Maaten & Hinton, 2008

Who Can See What, When:

- Training data: Available to developers
- Validation data: Available during development
- Test data: Restricted access
- Test results: Run by independent party

Why Governance Matters:

- Repeated test usage overfitting
- Public leaderboards incentivize gaming
- Business decisions need unbiased estimates

Practical Process:

1. Create test set at project start
2. Lock it away (separate repo/access)
3. Develop using train + validation only
4. Run test evaluation once for go/no-go
5. Document results, don't iterate

For LLMs/RAG:

- "Golden set" of Q/A pairs
- Expert-validated answers
- Versioned and maintained

Executive Mandate

Require documented evaluation governance for every AI project.

[illegible]

Production Monitoring: Because Deployment Is Just The Beginning

Model monitoring is product monitoring

van der Maaten & Hinton, 2008

What to Monitor:

- Input drift: Are queries changing?
- Output quality: Hallucination rate, refusals
- Retrieval health: Empty results, low scores
- Latency: Response time percentiles
- Cost: Tokens used, API spend
- Security: Injection attempts, data exposure

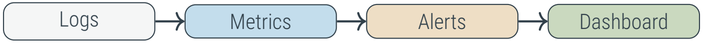
Alert Thresholds:

- Warning: 10% drop in user satisfaction
- Critical: Retrieval failure rate > 5%
- Critical: PII detected in outputs
- Warning: Latency p95 > 5 seconds

Feedback Integration:

- User thumbs up/down
- Escalation to human
- Correction submissions

The Monitoring Stack:



Act II: Complete Summary

What executives now understand about how AI works

van der Maaten & Hinton, 2008

Part A — ML Foundations:

- Supervised/unsupervised/RL taxonomy
- Classical methods still valuable
- Right tool for right problem

Part B — Deep Learning:

- Learned representations are key
- Optimization is about generalization
- Failure modes are predictable

Part C — Transformers:

- Embeddings enable semantic search
- Attention enables context understanding
- Context windows have trade-offs

Part D — RAG Systems:

- RAG grounds LLMs in your data
- Multiple variants for different needs
- Permissions and audit are mandatory

Part E — Evaluation:

- Benchmarks ≠ your task
- Test set governance prevents self-deception
- Production monitoring is continuous

The Meta-Lesson

Summary

- Summary point 1
- Summary point 2
- Summary point 3
- Summary point 4

Thank you!