From Words to Vectors to Answers

Embeddings, Transformers, RAG, and Agents in Finance

Day 6 Mini-Lecture, 60 Minutes

Discovery: Make a Computer Compare Two Sentences

A spreadsheet can sort numbers instantly, but it cannot tell that two sentences mean the same thing.

- "Profits rose sharply" and "Earnings jumped" share no words in common.
- Yet any analyst sees they say nearly the same thing.

Hint

Computers are superb at arithmetic on numbers. What would you have to turn each sentence into first?

Words as Vectors

Before computers can process language, text must become numbers. Every word, sentence, or document is mapped to a point in a high-dimensional numeric space. **The core idea: map words, sentences, or documents to points in space**

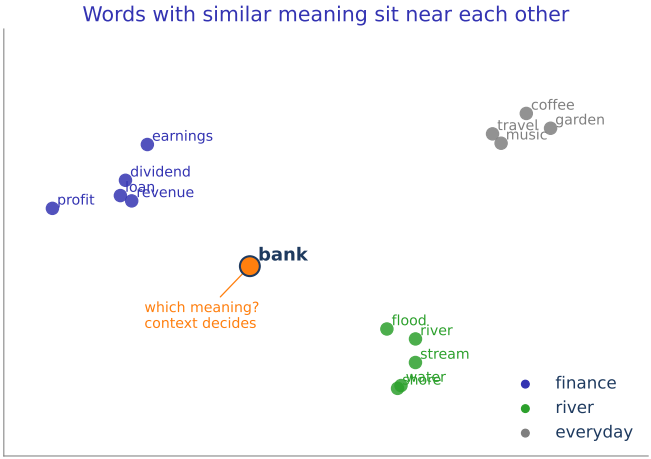
- A **vector** is just an ordered list of numbers: "bank" $\rightarrow$ [0.2, -0.7, 1.1, ...]. Here the list has 384 entries, so we say it has 384 **dimensions**.
- Words with similar meanings end up **near each other** in this high-dimensional space
- Meaning is now measurable: two vectors that are close = similar topic
- Distance in space reflects distance in meaning

Why it matters

Text is not naturally numeric. Embeddings convert language into geometry that any algorithm can compute on, enabling clustering, ranking, and retrieval at scale.

Embeddings turn meaning into geometry

Embedding Space: Meaning Becomes Geometry



Words with similar meaning cluster; an ambiguous word like “bank” sits between regions

Discovery: Finish the Word Analogy

Suppose each word is a point in space, and we can add and subtract those points.

- Paris relates to France the way Tokyo relates to which country? Write it down.
- Now commit to one word: king is to man as queen is to which word?
- Could plain vector arithmetic land on your answer on its own?

Hold your two answers; the next slide does this with real word vectors

The Famous Example: king – man + woman = queen

Arithmetic in embedding space reflects real-world relationships

- The model learned: "royalty" lives in one direction of space, "gender" in another
- Subtracting the "man" vector removes the male component
- Adding the "woman" vector shifts the result to the female side of the gender axis
- The resulting vector lands nearest to "queen" in the vocabulary
- This emerges from training on billions of sentences: no one programmed it

Key insight

The arithmetic works because the model has compressed the world's text into a geometry where relationships are **preserved as directions**. Similar relationships = parallel vectors.

Vector arithmetic in embedding space reflects semantic relationships learned from text

Worked Example: king minus man plus woman

Tiny two-dimensional vectors on a gender axis (x) and a royalty axis (y).

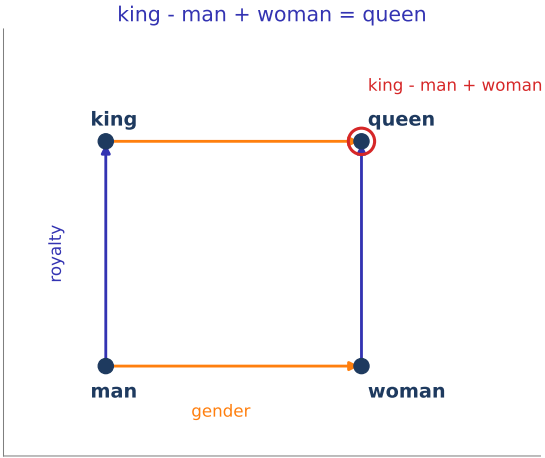
- man $= [0, 0]$, woman $= [2, 0]$, king $= [0, 2]$.
- Compute king $-$ man $+$ woman $= [0, 2] - [0, 0] + [2, 0] = [2, 2]$.
- The vocabulary word nearest $[2, 2]$ is queen.

What happened

Subtracting man removed "male", adding woman supplied "female", and the royalty component stayed. In real embeddings this holds only approximately, and the three input words are excluded when searching for the nearest word.

On toy vectors, king minus man plus woman equals $[2, 2]$, which is queen

Vector Arithmetic in Embedding Space



Relationships are preserved as directions: king minus man plus woman lands on queen

Discovery: Group These Companies Without Reading

You hold transcripts from two banks, two chipmakers, and an oil producer.

- Suppose each transcript is already a point in space.
- You want to group "similar" companies, but you are not allowed to read them.

Discuss with your neighbour (1 minute)

If each call is a point, what would make two of those points count as "close"?

Finance Analogy: Earnings Calls as Vectors

Two earnings calls discussing the same topics land nearby in embedding space

High similarity (same sector):

- FirstBank + RegionalBank: NIM, CET1, NPL ratios, provision for credit losses
- TechCore + ChipMaker: capex, AI accelerators, GPU, data center expansion

Low similarity (cross-sector):

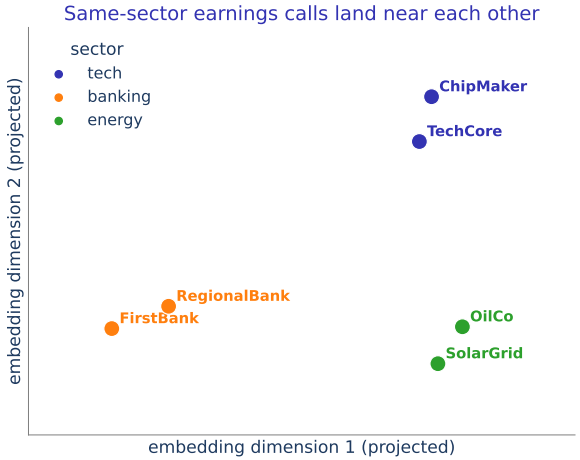
- TechCore + FirstBank: very different vocabulary, very different metrics

Applications:

- Find all filings discussing credit risk without keyword rules
- Detect strategy shifts between Q3 and Q4
- Cluster competitors by business model

Embedding space captures domain similarity; nearby vectors discuss similar business topics

Earnings Calls Cluster by Sector



Same-sector calls land near each other in embedding space

Discovery: Score How Alike Two Things Are

You want a single number for "how similar" two documents are.

- Pick the range for your score: should identical mean 1 and unrelated mean 0?
- A 3-sentence note and a 10-page report cover the very same topic. Same score, or not?
- Write the number you would give that pair.

Keep your scale and number; the next slide defines the standard measure

Cosine Similarity: Measuring Closeness

How do we measure whether two vectors point in the same direction?

- **Cosine similarity**: measures the angle between two vectors
- Small angle (vectors point in same direction): score close to 1.0
- Large angle (vectors point in different directions): score close to 0.0
- Cosine ranges from -1 to 1; for many sentence-embedding models scores mostly land in 0 to 1, but negatives can occur

Why cosine, not plain Euclidean distance?

- A 3-sentence paragraph and a 10-sentence article on the same topic should score high
- Their vectors differ in **length** (magnitude) but point in the **same direction**
- Cosine ignores length entirely: it measures direction only

Cosine similarity is **scale-invariant**: length encodes quantity of text, direction encodes meaning.

Cosine similarity measures semantic direction, not document length

Worked Example: Cosine Similarity by Hand

Two short vectors: $a = [1, 1]$ and $b = [2, 2]$.

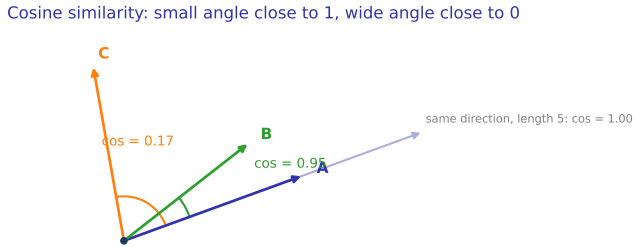
- Dot product: $1 \times 2 + 1 \times 2 = 4$.
- Lengths: $|a| = \sqrt{1+1} = \sqrt{2}$ and $|b| = \sqrt{4+4} = \sqrt{8}$, so $|a|\,|b| = \sqrt{16} = 4$.
- Cosine: $4/4 = 1.0$. Same direction, maximal similarity, even though b is longer.

Contrast

For $a = [1, 0]$ and $c = [0, 1]$ the dot product is 0, so cos = 0: unrelated directions.

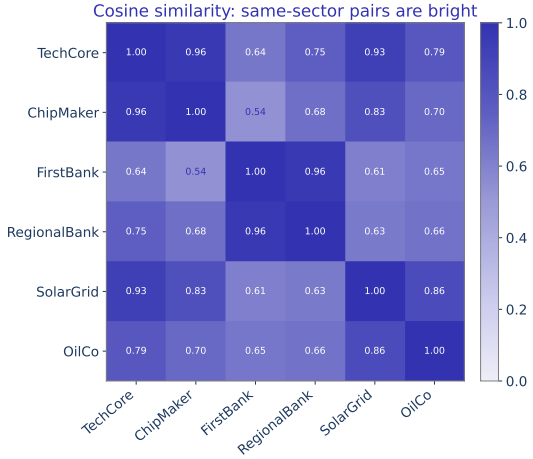
Cosine of $[1, 1]$ and $[2, 2]$ is 1.0; length does not matter, direction does

Cosine Similarity: Angle, Not Length



A small angle scores near 1.0; vector length does not change the score

A 6x6 Cosine Similarity Matrix



Same-sector pairs are bright: this is the matrix you build in Notebook A

Discovery: What Does "Bank" Mean Here?

Read two sentences and notice how you decide.

- "The bank was closed due to flooding."
- "The bank reported losses this quarter."

Discuss with your neighbour (1 minute)

Which other word in each sentence told you what "bank" meant? How did you know where to look?

Attention: Which Words Matter for This Word?

Attention answers the question: when processing a word, which other words in the sentence are most relevant to its meaning? **Language is context-dependent: one word can mean different things**

- "The bank was closed" – financial institution or riverbank?
- Attention lets the model look at **all other words** before deciding what a word means
- "The bank was closed due to flooding" – "flooding" gets high attention for "bank"
- "The bank reported losses this quarter" – "reported" and "losses" get high attention

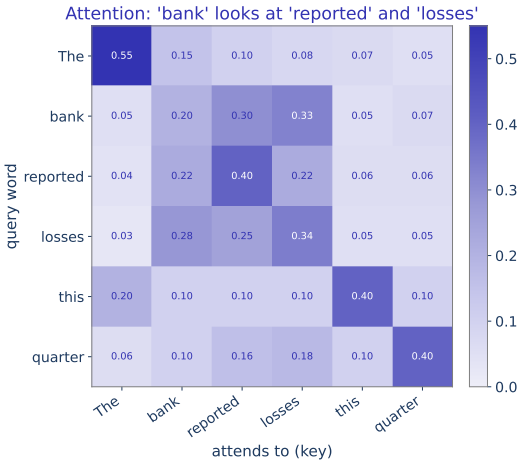
Mechanistically:

- Every word scores every other word by similarity; those scores, normalised, become the attention weights
- Relevant words contribute more to the final representation of that word
- This happens in parallel for every word in the sequence simultaneously

---

Attention replaces fixed word meanings with context-sensitive meanings

Attention Weights for One Sentence



Processing “bank”, the model puts most weight on “reported” and “losses”

Discovery: One Vector for a Whole Sentence

You can now turn each word into a context-aware vector. But search needs one vector per sentence.

- A 12-word sentence gives you 12 word vectors. Search wants a single summary.
- Simply averaging them throws away which words mattered most.

Hint

What if the model reserved one special slot whose only job was to absorb the whole sentence?

The Encoder: Reading and Building Representations

The encoder processes a full input sequence and produces rich vector representations

Input: tokens (the word-pieces the text is split into)

- "TechCore" "reported" "record" "cloud" "revenue"

Layer-by-layer processing:

- Layer 1: basic word-level relations
- Layer 2: phrase structure
- Layer 3: sentence-level semantics

Output: one context-aware vector per token

- Each output vector "knows about" its neighbours
- A special marker slot, `[CLS]`, whose final vector is used as the whole-sentence summary
- This summary vector is the **sentence embedding**

paraphrase-MiniLM-L3-v2

3 encoder layers, 384 dimensions. Runs locally in Colab, no API key required.

The encoder builds a context-aware representation of the entire input sequence

Discovery: One Analyst or a Team?

You must summarise a dense earnings call covering many themes at once.

- Option A: one analyst reads it and tracks everything alone.
- Option B: four analysts read in parallel, each watching one thing (numbers, tone, names, references).
- Which setup is more likely to catch every kind of pattern?

Choose one; the next slide builds the team version inside the model

Multi-Head Attention: Specialised Readers

One attention mechanism is useful; several in parallel are much better

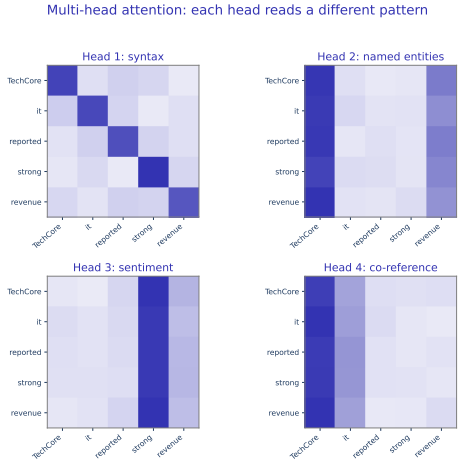
- **Each "head" attends to a different aspect of the sentence simultaneously:**
  - Head 1: syntax and grammatical agreement
  - Head 2: named entities (company names, numbers, dates)
  - Head 3: sentiment shifts (positive, cautious, negative language)
  - Head 4: co-reference ("it" refers back to "TechCore")
- All heads run in parallel; their outputs are concatenated and projected
- Result: one vector that encodes all these aspects at once

Analogy

Like having four analysts reading the same earnings call, each highlighting different patterns, then merging their notes into one combined summary.

Multiple heads specialise in different patterns (a useful caricature; real heads are more entangled)

Multi-Head Attention: Four Specialised Readers



Each head attends to a different pattern: syntax, entities, sentiment, co-reference

Discovery: Read It All, or Read Left to Right?

Two ways to build a language model:

- Model 1 sees the whole sentence at once and fills in a blanked-out word.
- Model 2 sees only the words so far and predicts the next one.
- Which fits understanding a filing? Which fits drafting one?

Pick one per task; the next slide is exactly these two designs

BERT vs GPT: Two Kinds of Transformer

BERT (Encoder-only)

- Reads the entire text at once (bidirectional)
- Trained to fill in masked tokens: "TechCore [MASK] record revenue"
- Sees left and right context simultaneously
- Best for: embeddings, classification, similarity search
- Our model today is BERT-family

GPT (Decoder-only)

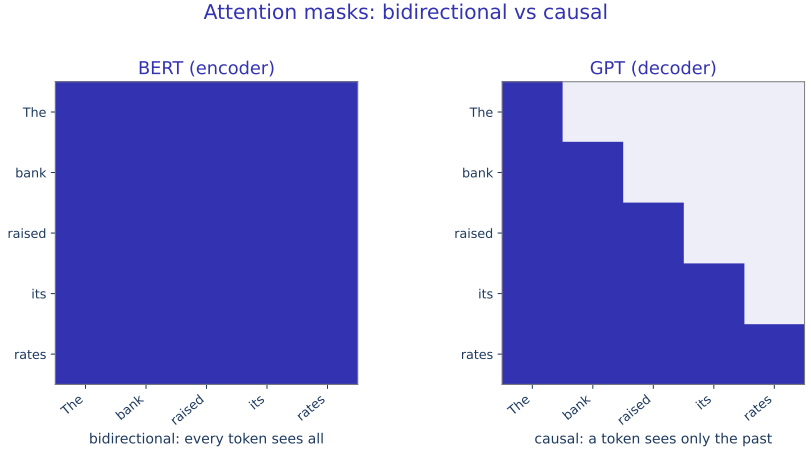
- Reads left to right only (causal/autoregressive)
- Trained to predict the next token
- Sees only past context, not future
- Best for: text generation, dialogue, completion
- ChatGPT, GPT-4 are decoder-only

Rule of thumb

To *understand* text (compute embeddings): use an encoder (BERT). To *generate* text: use a decoder (GPT). RAG uses both: BERT retrieves, GPT answers.

BERT reads bidirectionally for understanding; GPT generates left-to-right token by token

BERT vs GPT: Bidirectional vs Causal



BERT sees every token at once; GPT sees only the tokens to its left

Discovery: Why Not Just One Big Layer?

To understand a sentence, a model stacks many processing layers instead of one wide one.

- Recognising letters, then words, then meaning feels like separate jobs.
- Could a single step really jump straight from raw characters to abstract meaning?

Hint

Think how you read: first the letters, then the words, then the point. Each stage builds on the last.

Why Deeper Models Learn Better Representations

Each layer refines the representation built by the layer below it

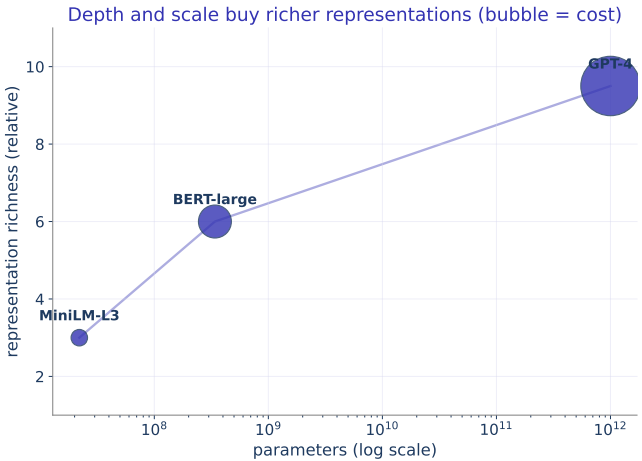
- Layer 1: raw character patterns, subword boundaries
- Layer 3: word-level semantics ("bank" resolved to financial context)
- Layer 6: phrase and clause-level structure
- Layer 9: entity relationships and co-reference
- Layer 12: abstract, task-relevant semantic concepts

Depth vs. cost trade-off (a *parameter* is one adjustable number the model learns; more parameters means more capacity):

- **paraphrase-MiniLM-L3-v2:** 3 layers, 22M parameters – fast, runs on CPU in seconds
- **BERT-large:** 24 layers, 340M parameters – richer, slower
- **GPT-4:** estimated 96+ layers – very rich, expensive, needs API

Depth composes low-level features into high-level concepts (the per-layer labels are illustrative; real layers are entangled)

Depth and Scale vs Representation Richness



Bigger models capture more; the bubble size shows the rising cost

Before We See: Large Language Models

Checkpoint: predict before we move on

- You now know transformers encode meaning into fixed-size vectors
- A small 3-layer model (MiniLM) produces 384-dimensional embeddings
- The encoder reads the full sentence and produces one context-aware vector per token

Discuss with your neighbour (2 minutes):

1. How does a language model generate a full sentence if it only predicts one token at a time?
2. Why can a model sound confident while being factually wrong (hallucination)?
3. What is the practical difference between a model that reads text (BERT) and one that writes it (GPT)?

Section 3 will answer each of these three questions.

Discovery: How Much Text to Learn Language?

A model learns grammar and facts purely by predicting the next word, over and over.

- Guess how many words of text it must read to get good: millions? billions? more?
- For comparison, a heavy reader might finish a few hundred books in a lifetime.
- Write your guess as a number before the next slide.

Keep your estimate; the next slide reveals the real scale

Pre-Training: Learning from Text at Scale

One task: predict the next word. Trained on everything

- GPT-3 was trained on about 300 billion tokens, filtered down from roughly 45 TB of raw web text: books, Wikipedia, web pages, code
- Only supervision signal: the text itself (no human labels needed)
- From this simple objective the model learns grammar, facts, reasoning, and style
- Scale changes qualitative behaviour: emergent abilities appear above certain sizes

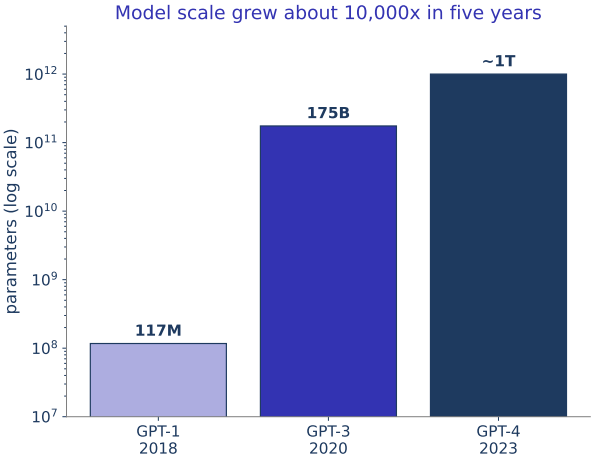
Scale milestones:

- GPT-1 (2018): 117 million parameters – single-task text understanding
- GPT-3 (2020): 175 billion parameters; few-shot learning (solving a task from a few examples in the prompt), code, and arithmetic emerge
- GPT-4 (2023): estimated ˜1 trillion parameters – multi-step reasoning, vision

---

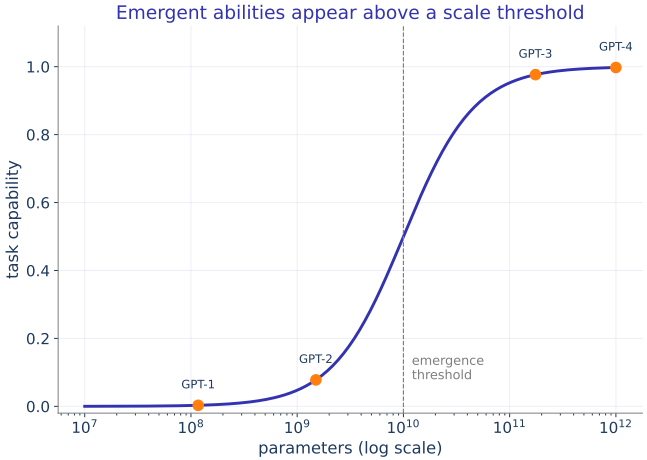
Self-supervised means the text is its own answer key: predict the hidden next word, no labels needed

Parameter Scale Across GPT Generations



Roughly 10,000 times more parameters in five years (log scale)

Emergent Abilities Appear With Scale



Some capabilities switch on only above a certain size threshold

Discovery: Reuse the Reader, or Start Over?

You already have a model that has read most of the internet. Now you need a finance-sentiment classifier.

- You have only 5,000 labelled earnings sentences, not billions.
- Training a fresh model from nothing on 5,000 examples would barely work.

Discuss with your neighbour (1 minute)

How could you keep everything the big model already knows and only teach it the new, narrow skill?

Fine-Tuning: Adapting a Pre-Trained Model

Take the pre-trained model; teach it a narrow skill with far less data

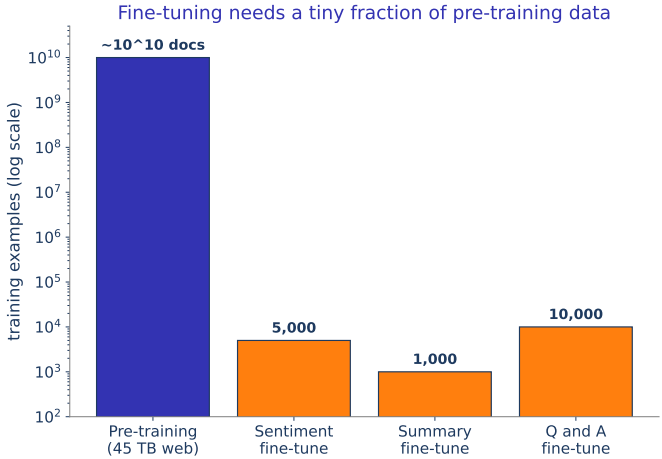
- **Sentiment analysis:** 5,000 labelled earnings calls; model learns financial tone
- **Summarisation:** 1,000 (report, summary) pairs; model learns concision
- **Q&A:** 10,000 (document, question, answer) triples; model learns reading comprehension

RLHF: aligning with human preferences

- Human raters compare and rank model outputs
- A reward model learns what humans prefer (helpful, honest, harmless)
- The LLM is updated via reinforcement learning to score higher on the reward model
- This is how ChatGPT, Claude, and Gemini are aligned

Fine-tuning is cheap relative to pre-training: one base model, many downstream tasks

Pre-Training Data vs Fine-Tuning Data



Fine-tuning needs a tiny fraction of the pre-training corpus (log scale)

Discovery: Guess the Next Word

Read this opening: "The quarterly earnings report showed revenue of".

- Write down the single word you expect to come next.
- Now give it a rough chance out of 100, and name one runner-up word.
- Where do you think the model keeps the "answer" it is about to give?

Hold your word and its odds; the next slide shows how the model picks one

GPT as a Prediction Engine

There is no stored database of answers: only a learned probability distribution

- Input: "The quarterly earnings report showed revenue of"
- Model scores every possible next token across its 100,000-token vocabulary
- Likely next tokens: "$", "3.4", "record", "a", "strong"
- Unlikely tokens: "elephant", "however", "if", "purple"
- Sample one token from this distribution (temperature controls randomness)
- Append the token; repeat from the beginning

What this means

Every response is **reconstructed token by token** from learned statistics. The model has no stored answers. This is why it can be wrong while sounding confident.

Generation is sampling from a learned distribution over the next token, not retrieval

Worked Example: Scoring the Next Token

Prompt: "The quarterly earnings report showed revenue of _____".

- The model assigns a probability to every word in its vocabulary.
- Plausible: "$" ≈ 0.22, "3.4" ≈ 0.18, "record" ≈ 0.16, "strong" ≈ 0.10.
- Implausible: "elephant" ≈ 0.005, "purple" ≈ 0.005.

Then it samples

It draws one token from this distribution, appends it, and repeats. It samples, it does not look up a stored answer.

Every token gets a probability; the model samples one and continues

Worked Example: Temperature Changes the Odds

Two candidate tokens with scores ("logits", the raw pre-probability numbers) 3.0 **and** 2.0**. Softmax turns scores into probabilities:** $p_i = e^{x_i/T} / \sum_j e^{x_j/T}$**.**

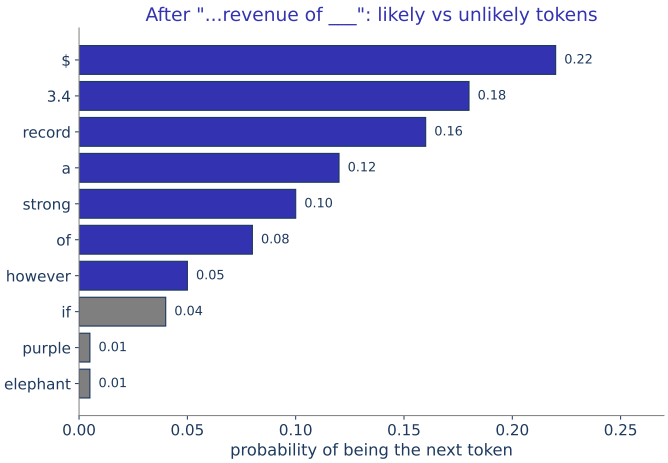
- At $T = 1$ (with $e \approx 2.72$): $e^3/(e^3 + e^2) = 20.1/27.5 \approx 0.73$, and the other is 0.27.
- At $T = 0.5$, divide the scores by 0.5 (so 6 and 4): $e^6/(e^6 + e^4) = 403/458 \approx 0.88$, and 0.12.
- At $T = 2$ (flatter): the gap narrows to about 0.62 and 0.38.

Intuition

Low temperature makes the model confident and repetitive; high temperature makes it adventurous and more random.

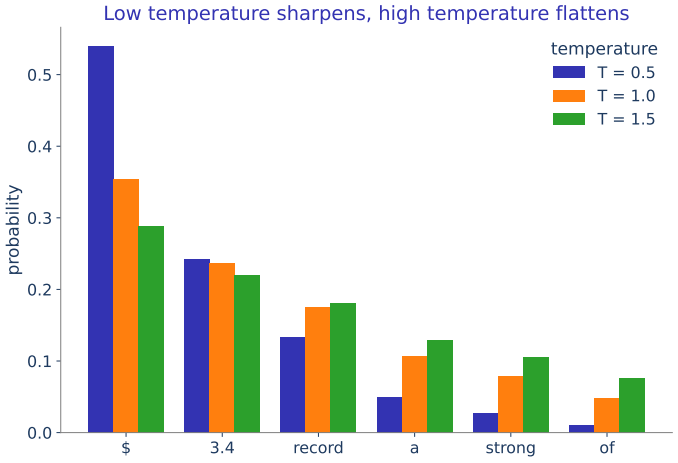
Temperature rescales the scores before sampling: low sharpens, high flattens

Next-Token Probability Distribution



The model scores every token, then samples one: it does not look up an answer

Temperature Reshapes the Distribution



Low temperature sharpens toward the top token; high temperature flattens the choices

Discovery: Writing With No Outline

The model can score the single next word. But you just asked it for a full paragraph.

- It does not plan the whole answer first, the way you might sketch bullet points.
- Somehow a coherent paragraph still appears, word after word.

Hint

If you can produce one good next word, what could you do with the sentence you have so far to get the word after that?

What ChatGPT Actually Does: Step by Step

When you send a message, the following happens:

1. **Tokenise:** split your text into word-pieces ("ChatGPT" → "Chat", "G", "PT"; spaces and punctuation included)
2. **Embed:** convert each token to an initial vector via a lookup table
3. **Transform:** pass through ~96 attention layers; context accumulates across all tokens
4. **Score:** final layer produces a probability distribution over all 100,000 tokens
5. **Sample:** pick one token (e.g. "The")
6. **Repeat:** append "The" to the input sequence; return to step 3
7. **Stop:** when a special `[EOS]` end-of-sequence token is sampled

---

ChatGPT builds the response one token at a time; it writes no outline first, and each token is chosen given all tokens so far

Discovery: Ask the Model About Yesterday

You ask a polished chatbot: "What was TechCore's capex in last quarter's call?"

- That call happened after the model finished training, so it was never in the data.
- The reply comes back fluent, specific, and confidently precise.
- It was never told this number. So where could that figure have come from?

Spot the danger before the next slide offers a fix

What is Retrieval-Augmented Generation?

Problem: LLMs have a knowledge cutoff and cannot access private documents

- GPT-4 does not know your Q3 2025 earnings calls – they were not in training data
- If you ask, the model will either say "I don't know" or **hallucinate** a plausible-sounding answer
- RAG solves this with a retrieval step before generation

Three components of RAG:

1. **Retrieval:** search a private document database using cosine similarity on embeddings
2. **Augmentation:** add the retrieved documents to the prompt
3. **Generation:** the LLM generates an answer grounded in the retrieved text

RAG combines search precision with language model fluency

Discovery: An Open-Book Exam

A model knows lots in general but nothing about your private filings.

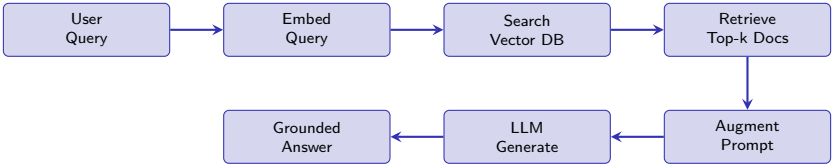
- Closed-book, it must answer your company-specific question from memory alone.
- Open-book, it may first pull the exact relevant passage, then answer.

Discuss with your neighbour (1 minute)

What steps would turn a closed-book model into one that always answers with the right page in front of it?

RAG Architecture

Five-step pipeline from question to grounded answer



The vector database contains **pre-embedded documents**: earnings calls, 10-K filings, research reports. The retrieval step uses **exactly the same cosine similarity** you computed in Notebook A. "Top-k" just means the k closest documents, for example the top 3.

The retrieval step grounds the LLM in actual documents rather than learned statistics

Discovery: Which Snippet Answers the Question?

A user asks: "What did TechCore say about capital spending last quarter?"

- Snippet A: TechCore's own remarks on data-center investment that quarter.
- Snippet B: a rival bank discussing its loan-loss provisions.
- Which snippet should the system hand to the model, and how would it decide automatically?

Pick one; the next slide walks the full query-to-answer path

Earnings Call Q&A: A Concrete Example

User query: "What did TechCore say about capital expenditure in Q3?"

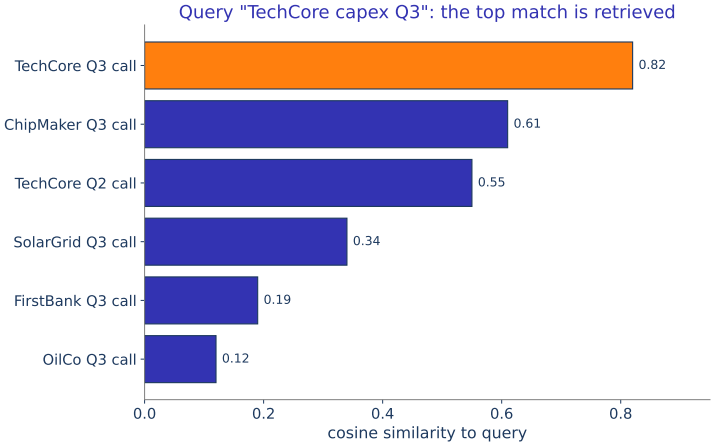
1. Embed the query: "TechCore capital expenditure Q3" $\rightarrow$ a 384-dimensional vector
2. Search the database of embedded earnings snippets (cosine similarity over all docs)
3. Retrieve the TechCore Q3 snippet (similarity score 0.82 – highest in the database)
4. Build the augmented prompt: *"Based on this passage: [TechCore Q3 text], answer: what did TechCore say about capex?"*
5. LLM responds: "TechCore reported capital expenditure of 2.1 billion dollars for data center expansion..."

Key point

The model did not memorise this number during training. It read the retrieved document and synthesised a fluent answer. The source document is verifiable.

The LLM acts as a reading-comprehension engine, not as a memory bank

Retrieval Ranks Documents by Similarity



The closest snippet (0.82) is retrieved and passed to the model

Discovery: A Confident, Wrong Answer

A model returns a precise capex figure for a quarter it was never trained on.

- The sentence is grammatical, specific, and sounds authoritative.
- An analyst pastes it straight into a client report.
- Why is a confident wrong number more dangerous here than a plain "I do not know"?

Spot why fluency is not truth before the next slide quantifies it

Risk: Hallucination Without RAG

When the LLM has no access to actual documents, it may fabricate

- Query: "What were TechCore's Q3 2025 capex figures?" (after its training cutoff)
- **Without RAG:** model generates a plausible-sounding but fabricated number
- Hallucination is confident, grammatically correct, and factually wrong

Why this is especially dangerous in finance:

- Investment decisions, compliance reports, and client communications may depend on these figures
- A wrong number delivered with confidence is more dangerous than "I do not know"

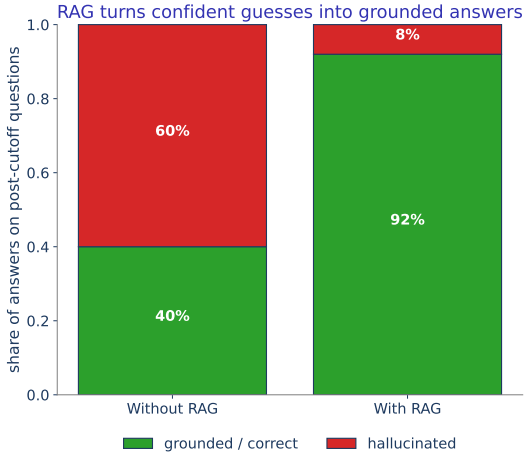
RAG as a safety mechanism:

- The answer is anchored to a retrieved source document that can be cited and verified
- The model is much less likely to contradict the passage it was given, though not impossible, so still verify

---

RAG mitigates hallucination by grounding generation in verifiable source documents

Hallucination: With vs Without RAG



Grounding answers in retrieved sources cuts confident fabrication

What You Will Build in Notebook A

Day 5 (Exercises 1–6): Earnings call similarity from scratch

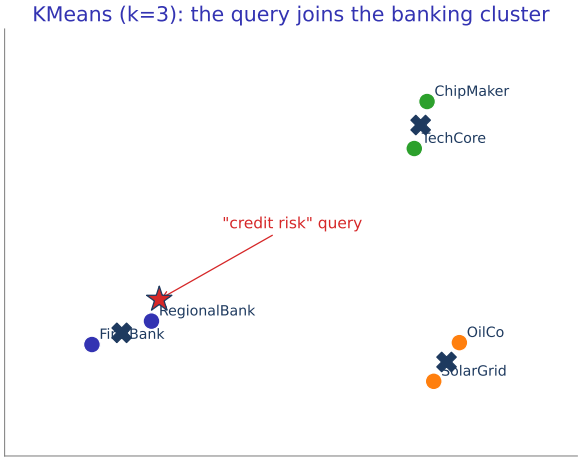
- Encode 6 synthetic earnings snippets (tech, banking, energy companies)
- Compute the full 6x6 cosine similarity matrix
- Find the most similar pair; rank by similarity to a query snippet; visualise the heatmap

Day 6 Phase 2 (Integration cell, 60 min): Mini-RAG pipeline

- Cluster the 6 snippets with KMeans using their embeddings
- Encode a "credit risk" query; compute which cluster centroid is most similar
- Train a LogisticRegression on 200 synthetic loan applicants
- Complete the synthesis: "Company X is in Cluster _____ because _____, the classifier predicts _____"

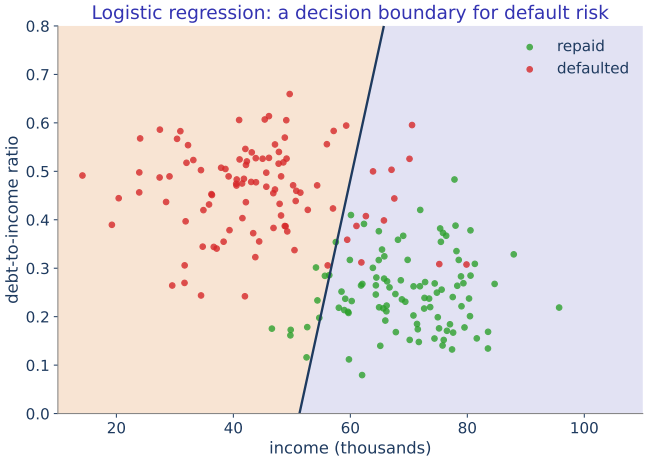
Notebook A connects every concept from this lecture to working, runnable Python code

Clustering the Snippets with KMeans



The credit-risk query joins the banking cluster (Notebook A, Phase 2)

Logistic Regression on Loan Applicants



A fitted boundary separates likely defaults from repayments (Notebook A)

Discovery: A Question One Prompt Cannot Answer

"What was Company X's revenue growth, and is it above the sector median?"

- The model alone has no live filing, no calculator, and no sector database.
- Answering needs several distinct steps, not one burst of text.

Hint

What if the model could call outside tools, look at each result, and decide its next move?

From Answering to Acting: What Is an Agent?

An LLM alone only predicts text. An agent wraps it in a loop with tools.

- **The loop:** think (plan the next step), act (call a tool), observe (read the result), then repeat until the task is done.
- **Tools** are ordinary functions: a document search, a price API, a Python calculator, a database query.
- The model decides *which* tool to call and *with what* inputs; the tools do the real work.

Finance example tools: a filings search, a returns calculator, a sector database.

An agent is an LLM plus tools plus a loop. There is nothing more magical than that.

Discovery: Plan the Steps Yourself

The task: "Find Company X's revenue growth and say whether it beats the sector median."

- You have a document search, a calculator, and a sector database available.
- No single one of them answers the question by itself.

Discuss with your neighbour (1 minute)

List the steps, in order, you would take to answer this. Which tool does each step use?

A Finance Agent, Step by Step

Task: "What was Company X's revenue growth, and is it above the sector median?"

1. Retrieve the latest annual report (tool: document search).
2. Extract the revenue figures (tool: parser).
3. Compute the growth rate (tool: calculator).
4. Look up the sector median (tool: database).
5. Answer, citing each source.

Why it helps: it decomposes a multi-step question that a single prompt cannot answer reliably.

Agents shine when a task genuinely needs several tools and steps, not for one-shot questions.

Discovery: Ten Steps, Each Usually Right

An agent chains 10 tool calls, and each one works correctly 90% of the time.

- The whole task only succeeds if every single step succeeds.
- Guess the chance all ten work in a row: about 90%, around 60%, or much less?
- Write your percentage before the next slide.

Hold your guess; the next slide does the multiplication

Why Agents Are Fragile

- **Errors compound.** A step that is 90% reliable, run ten times, is only $0.9^{10} \approx 0.35$ reliable end to end.
- **Cost and latency.** Every step is another model call, so long chains are slow and expensive.
- **No guarantee of stopping or of truth.** An agent can loop, invent a tool result, or take an unsafe action.

When not to use one: if a single retrieval or a plain script solves the task, prefer that.

Reserve agents for genuinely multi-step, tool-using tasks, and put guardrails around any real-world action.

Worked Example: Why Agents Are Fragile

An agent chains 10 tool calls; each one is 90% reliable on its own.

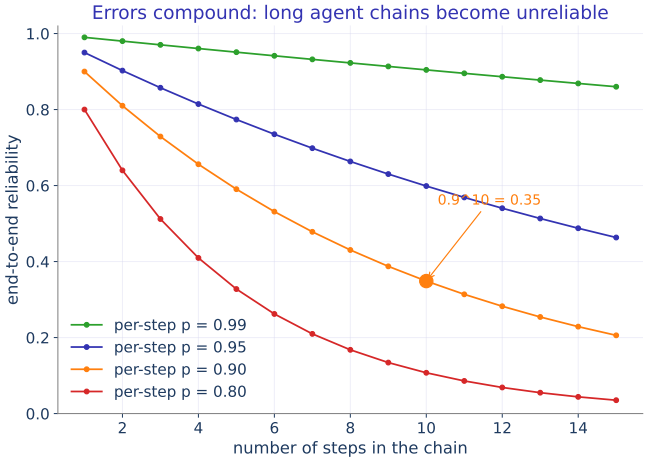
- The whole task succeeds only if every step succeeds.
- End-to-end reliability: $0.9^{10} = 0.349$, about 35%.
- Even at 99% per step, ten steps give $0.99^{10} \approx 0.90$, so 1 run in 10 still fails.

The takeaway

Errors compound. Keep tool chains short, and put a check after any step that matters.

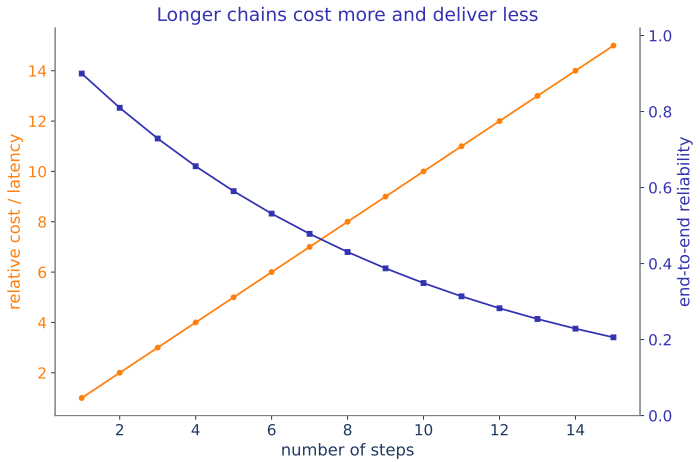
Ten steps at 90% each is only $0.9^{10} \approx 0.35$ reliable end to end

Agent Reliability Decays as Steps Grow



0.9 reliable per step over ten steps is only about 0.35 reliable end to end

Longer Chains: More Cost, Less Reliability



Cost and latency rise with each step while end-to-end reliability falls

Discovery: “It Sounded Completely Sure”

A junior colleague says: “The model is fluent and confident, so I trust its numbers.”

- The output is articulate, well organised, and never hesitates.
- It also has no access to events after its training and cannot truly calculate.
- What is wrong with treating fluency and confidence as proof of correctness?

Spot the mistaken assumption before the next slide lists the real limits

The Honest Limits of Large Language Models

- **Hallucination:** fluent text is not verified truth. A model can state false facts with total confidence.
- **Knowledge cutoff:** it does not know events after training unless you supply them, which is exactly why RAG matters.
- **No guaranteed reasoning:** this is pattern completion, not a calculator. Check anything that must be exact.
- **Cost and bias:** large models are expensive to run and inherit the biases in their training data.

Knowing these limits is not pessimism. It is the difference between a practitioner and a fan.

Using the Frontier Responsibly in Finance

- **Ground every answer that matters** in retrieved, citable sources (RAG), never the model's memory alone.
- **Keep a human in the loop** for any decision that involves money or compliance.
- **Measure honestly.** The evaluation traps from Day 2 apply double to a flashy demo, so beware gaming the metric.

The mature stance: powerful tools, narrow guarantees, used with sources and oversight.

The frontier is most useful precisely when you respect its limits.