LLM / AI OVERVIEW

Contents:

- Why this process allows meaning to emerge, not just memorisation
- How this step lets the model work with long, complex sentences
- How it uses the updated input to guess the next likely word
- Why it doesn't just pick the most common word every time (sampling)
- How randomness (temperature) affects creativity vs accuracy
- How grammar, structure, and style are maintained across long answers
- How it can "remember" what was said earlier in the same conversation

Section 6: How are LMMs Controlled

What stops it from saying dangerous, offensive, or false things

- What safety prompts and rules are built into its instructions
- How humans rate outputs during training to guide good behaviour (RLHF)
- Why some answers are blocked, softened, or redirected
- What limitations still exist - and why "bad output" is still possible

Section 7: What the Best Systems Can Do Today

How different versions of these models compare - and what they're good at

- A look at GPT-4, Claude, Gemini, Mistral, LLaMA, and others
- Which models are good at what (text, reasoning, coding, safety)
- Why newer models are better: more data, smarter training, better tuning
- Why "open source" and "closed" models differ in control and capability

Section 8: What's Coming Next

How future models may improve and what we're still trying to solve

- Giving models memory that lasts beyond one chat
- Combining language with vision, audio, code (multimodal)
- Making models run faster and cheaper (efficiency)
- Exploring new brain-like systems (beyond transformers)
- Agents – already here today and how they work

Section 9: How Vision and Language Models Are Starting to Merge Together

- How language and vision models started from different methods
- How images are broken into parts like tokens (patches)
- How transformers process images and words with similar structures
- How combined models link words and images into shared number spaces
- How multi-modal models can describe images, answer questions, or generate new ones
- How this fusion may lead toward systems that combine multiple types of knowledge like the human brain

Section 10: Multi-Modal Models as the Potential Future Basis for Robotics

- Why real-world robots need vision, language, planning, and adaptation together
- How multi-modal models combine vision, language, and reasoning into one system
- How real-time sensor data can feed directly into transformer-based models
- How motor control and physical movement are layered onto multi-modal processing
- How companies like Wayve, NVIDIA, and DeepMind are developing these full-stack systems
- Why this approach may finally make adaptive smart robots practical in the near future

Synopsis

Large Language Models (LLMs) are software systems that run on powerful hardware, especially GPU clusters ***that can process vast amounts of data in parallel***. These systems require large-scale infrastructure, including energy, cooling, and memory, to operate.

LLMs convert text ***into lists of numbers*** (vectors) using a learned table. Once in number form, these vectors pass through many mathematical layers. Each layer transforms the input based on patterns learned during earlier training. The system does not use logic or rules - it works entirely by processing number patterns.

Training involves feeding the model billions of text samples and having it guess the next word. Each wrong guess triggers a small adjustment to internal values (weights). Over time, this process forms a complex statistical pattern of language use. The model stores no facts or meaning - only numeric relationships shaped by past examples.

When prompted, the model produces output one 'token', (word or part of), at a time. It uses past input and output to guide each next prediction. There is no long-term memory, awareness, planning, or understanding - only mathematical prediction based on training patterns. Text that is input during training is not captured in an sense as text. The LLM can only deduce text from its mathematical patterns. Some very common ideas and text that the model 'sees' many times becomes 'known' within these patterns but most of the text it is trained on is lost.

Safety controls are added through training and filtering. These include human feedback, blocked responses, and refusals. ***These systems reduce but do not eliminate errors***, bias, or harmful outputs.

Some models now ***process both text and images*** by treating image parts like text tokens. This allows them to match captions to pictures or describe visual content. These systems still rely on token prediction and do not "see" or "understand" in a human sense.

Agents are small software programs that call LLMs repeatedly to complete simple tasks. They do not think or plan. They follow a loop: send a prompt, get a reply, decide what to do next, repeat. Some can use tools between steps (e.g. search engines), but current agents are limited and experimental. They do not integrate complex goals or operate independently.

LLMs do not understand, reason, or generalise. ***They are not intelligent.*** They generate output by reproducing learned patterns of text, using statistical prediction at scale. The user may be confused by the LLM into thinking they are doing some form of 'thinking', but this confusion arises because the LLM is very good at organising language and outputting sensible comments. The companies creating these LLMs also program them to mirror a user's words and phrases, this also creates the impression of 'thinking'.

Section 1: What are LLMs built From

How the physical machines and chips make an LLM possible

1. Why Hardware Matters

Large language models (LLMs) like this one are software - lines of code - but they only work because they run on very **powerful physical machines**. Your laptop or phone can run many programs, but it **can't handle** what LLMs do. That's because these models need to do **billions or even trillions of number calculations** - every time they answer a single question.

To make this possible, they need a specific kind of machine built for **very fast, very large-scale number crunching**.

2. The Special Chips That Do the Work

Most computers use something called a **CPU** (Central Processing Unit). It's good at doing one task (sequentially) at a time - very quickly (GHz). But LLMs need something different: they need to do **millions of small tasks all at once**.

That's why they use **GPUs** (Graphics Processing Units). These chips were originally made for drawing images and video on VDU screens, but they turned out to be perfect for running LLMs.

GPU = a chip that can do thousands of calculations at the same time (parallel processing). GPUs break up a big problem (like understanding a sentence) into **tiny parts**, then solve all those parts **at once**.

3. What the GPU Is Actually Doing

Let's say you type a sentence: "Why is the sky blue?" Behind the scenes, the LLM:

- Breaks the sentence into small pieces (tokens)
- Turns each piece into a list of numbers (vectors)
- Passes these numbers through **layers of mathematical steps**
- Compares, updates, and refines the numbers again and again

Each of these steps is **pure math** - adding, multiplying, comparing, and updating numbers. And this happens **billions of times**, just to produce one short response.

The GPU is the engine that makes this fast enough to be useful - it's **not thinking**, it's just **processing numbers very, very efficiently**.

4. The Bigger Setup

One GPU isn't enough to run a big model. Instead, companies build **GPU clusters** - racks of machines, each with 8–16 GPUs inside, working together. Then they connect many of these racks to form a **data centre**.

Each cluster needs:

- **Memory**: to store huge lists of numbers and results
- **Cooling systems**: to prevent overheating
- **Power supplies**: some of these clusters use **as much electricity as small towns**
- **Network connections**: to let all parts talk quickly

This is why LLMs can cost **millions of dollars to train**, and why using them (like we're doing now) still costs real money per second.

5. The Scale of Modern LLM Hardware

Big companies like OpenAI, Anthropic, Google, Meta, and Microsoft build **AI supercomputers** - data centres filled with thousands of connected GPUs.

- A single **training run** for GPT-4 may use **tens of thousands of GPUs**, running for **weeks**.
- These setups are built in **cooled buildings**, with direct access to **high-voltage power grids** and **fibre-optic internet**.

An LLM might need as much hardware space as a football field to run at full speed. And that's just the beginning - many models are growing in size and needing even more computing power.

6. Summary of What It's Built From:

Even though the LLM seems like it's "just software," it runs on one of the **most powerful physical computing systems** humans have ever built.

- **GPUs** make fast, parallel math possible
- **Data centres** keep everything powered and cooled
- The system works by **doing huge numbers of calculations**, every second, for every question

But here's something important:

The software doesn't contain rules or logic like "if this, then that." It doesn't understand your question like a traditional program would.

Instead:

- It sets up a **giant structure of numbers** (called a matrix)
- Each word becomes a long **list of numbers** (a vector)
- The software's job is to **organise and apply** a massive set of calculations to those numbers

The output - like this paragraph you're reading - isn't created by logic trees or decisions. It comes from **patterns of numbers interacting**, based on **what worked well during training**.

- The LLM doesn't "think" in steps - it flows through math that **looks like reasoning**, because the **pattern** it learned *resembles* how humans reason. We'll explain how that works more clearly in later sections.

Section 2: How Words Become Numbers for GPU Processing

How the system turns language into something it can work with

1. The Model Starts with Nothing

When a large language model (LLM) is first created, it knows absolutely nothing.

- It has no rules
- No facts
- No grammar
- No understanding of language at all

All it has is a **fixed** structure - like a big empty brain full of knobs and dials, all set randomly. This structure is made of layers and connections between them, but every connection starts with a random numbers.

2. Why the Model Needs Numbers, Not Words

Computers, including LLMs, **don't understand words** the way humans do. A word like "sky" is meaningful to us, but to a machine it's just a bit of text - and that's useless without conversion.

What computers *can* work with are **numbers**. So before a large language model can do anything - like answer a question or write a sentence - it needs to **turn your words into numbers**.

But not just one number per word. Each word gets turned into a **long list of numbers** that together represent its **meaning and context**. This list is called a .

How many numbers in a vector?
A typical vector has 512 to 32,000 numbers, depending on model size. The number is chosen by the designer researchers before training. These numbers don't directly mean anything individually. Instead, they form a flexible space where relationships between words can be represented. The model learns which parts of the list should carry which kinds of information during training.

3. Breaking Language into Smaller Pieces

Before the model turns anything into numbers, it **splits your sentence into smaller parts**. These are called **tokens**. A token is usually a word, part of a word, or even punctuation.

For example:

- "cat" is one token
- "running" might be split into "run" and "##ning"
- "don't" might be split into "do" and "n't"

This helps the model handle words it's never seen before, or rare spellings, or parts of longer compound words:

- So Step 1 = *break the sentence into tokens.*

4. Turning Tokens into Vectors (Lists of Numbers)

Each token is then **converted into a vector** - a long list of numbers.

Example:

"cat" → `[0.21, -0.44, 0.05, 0.88, …]`

This vector might have **hundreds or even thousands** of numbers in it. Each number captures **some part of how the model sees the token** - its general meaning, position, or how it's typically used.
But no human wrote those numbers down.

5. Where Do These Numbers Come From?

The model **learned** them during training.

- When the model was trained on huge amounts of text, it adjusted the numbers for each token so they helped it guess what comes next in a sentence.
- This list of numbers (vector) was gradually shaped so the model could spot patterns and meaning.

All these vectors are stored in a huge lookup table called the **embedding matrix**.

You can think of it like a giant spreadsheet:

- Row = token
- Columns = numbers in the vector
- Lookup: "What's the vector for 'cat'?" → go to that row and grab all the numbers

6. Why This Works So Well

This system of token-to-vector conversion is what gives **LLMs their superpower**: it turns words into numbers that capture how words relate to each other, so the model can combine and compare these number patterns to find meaning and context as it processes language.

- Words like "cat", "dog", and "pet" will have **similar vectors**, so the model knows they're related.
- Words like "bank" will shift their meaning **based on linked words** like "money" vs "river" - because the final meaning is updated later by the ***attention process*** (coming in Section 5).

The subsequent *vector* is the **starting point** for the model's entire reasoning process.

7. What Happens Next?

Once every token has been turned into a vector, the model stops thinking about "words." From this point forward, the model works entirely with:

- **Vectors** (lists of numbers)
- **Matrices** (tables of many vectors)

This is what the model "sees": A sentence becomes a matrix of numbers - think off a **3D spreadsheet** - and **that** is what goes into the core *thinking* system (called the ***transformer***, coming soon). The language is gone. Now it's just **patterns in number-space**.

Summary

To do anything useful, the LLM must first **turn your words into numbers** - but in a smart way.

- Words or parts of words become tokens
- Tokens become *vectors* - long lists of numbers
- These vectors are stored in a big 3D lookup memory table
- These numbers are learned through training - not written by a programmer
- This number matrix lets the system compare, relate, and understand

Without this step, the model is blind. This is the **foundation** of all further pattern matching.

Section 3: How the System Teaches Itself

How the model learns by guessing words and correcting itself over time

1. The Training Process:

The training stage is programmed, it is a script of instructions telling the hardware to perform calculations (using PyTorch or TensorFlow libraries).

The training program includes a loop that iteratively feeds data, computes loss, back-propagates gradients, and updates weights.

2. The Task: Predict the Next Word

The whole training process is built on a **simple challenge**: Show the model part of a sentence, and ask it to **guess what word comes next**.

For example:

- Show it: “The sky is full of”
- The correct next word might be: “stars”
- The model guesses: “clouds”

It’s **wrong** - but that’s okay. Here’s the key: the model uses that mistake to **improve itself**.

3. Learning by Mistake

Each time the model makes a guess, it compares:

- What it *thought* the next word would be
- What the *real* next word actually is

Then it slightly adjusts its internal settings - the values stored in its **weights** - to be a little better next time.
It’s like turning a million tiny dials by a tiny amount, based on how far off the guess was.
This process of adjusting the weights is called **backpropagation** (we’ll look at this later), and it’s guided by a method called **gradient descent** - both are just ways to slowly improve the model’s accuracy over time.

What matters is:

- Every mistake helps the model get **slightly smarter**. And it does this **billions of times**.

4. Learning at Massive Scale

This guessing game isn’t just done once or twice - it’s done:

- With billions of sentences
- For weeks or months
- On thousands of GPUs

The model reads books, websites, conversations - anything that humans have written. It keeps guessing and adjusting, over and over. From this, it gradually figures out:

- Grammar
- Vocabulary
- Word relationships
- Sentence structure
- Even patterns of logic and reasoning

But here's the strange part: no one tells it the rules. It figures them out on its own, by noticing what usually comes next.

5. Where the Learning Helps:

The model doesn't write down rules like "a noun goes after a determiner." Instead, the learning is baked into the **weights** - millions or even **billions of number values** stored inside the model.

These weights affect how each input vector (a list of numbers from a word) gets changed as it moves through the layers of the model.

These weights *are* the learned knowledge - not words, not sentences, just finely tuned numbers that reshape inputs in smart ways, they form a learnt pattern.

6. Pattern Pressure – Creates Output

When you give the model a sentence, it breaks it into small pieces (tokens). Each token is converted into a long list of numbers that represent what the model learned about that word during training. The system then compares every word with every other word to see which ones relate. It does this by multiplying the numbers in their lists to get a score for how closely they connect.

These scores are used to update the numbers for each word, blending its meaning with related words nearby. This is repeated many times through many layers.

"Pattern pressure" means that these adjustments keep pulling the numbers towards combinations that match patterns the model saw often during training. If "sky" and "blue" often appeared together, the numbers naturally shift to favour that pair.

7. At The End of the Training:

Once enough guessing and adjusting has happened - and the model is good at predicting - training stops.

At that point, the model:

- Doesn't keep learning on its own
- Doesn't need new information to function
- Can generate useful, intelligent responses using only what it has already learned

You're using the output of one of those trained models now. Everything you see in this tutorial is based on what was stored during that massive offline training phase.

8. Training vs Using:

Training:

- Happens once
- Requires supercomputers and weeks of time
- Costs millions of pounds or dollars
- Involves massive data sets

Using:

- Happens every time you ask a question

- Runs on a smaller system using the frozen model of weights across the 3D matrix
- No new learning - just applying what was already trained

So when I answer you now, I'm not learning anything new - I'm just applying what I already learned during my "education."

Summary

Training an LLM is like teaching it by showing it the world - one word at a time, and then getting it to tune itself.

- It starts off knowing nothing
- It reads billions of text samples
- It stores learning as patterns in number values (weights)
- It improves by guessing and correcting itself
- This Back-Propagation allows the matrix to improve its patterns
- Once trained, it can generate complex language, but it doesn't learn new things unless retrained
- This final matrix is fixed, and can be used to create another copy running on a new computer system

Section 4 Overview: How Knowledge Is Stored as Numbers in the Matrix

How everything the model has learned is captured and saved - not as facts, but as patterns of numbers

Key Concepts to Build Up:

- The model doesn't remember quotes or facts like a hard drive - it remembers **statistical patterns**.
- The main thing it saves is a massive set of **numbers called weights**, which determine how input tokens get transformed.
- These weights form layers of connected numbers - like filters - that reshape inputs based on learned relationships.
- These weights live inside 3D **matrices** - large layered spreadsheets full of values - creating each layer of the model.
- The weights don't store words or meanings directly, but **how to process and combine token-vectors** based on past training.
- The model also stores **token embeddings** - pre-learned vector values for each common token.
- Together, this lets the model respond to new prompts by applying what it learned - even if it doesn't remember specific past examples.

Section 4 Detail:

How the system remembers patterns - not words - using millions of learned numbers

1. What Is Saved After Training?

Once the model finishes learning, it doesn't store:

- Sentences
- Facts
- Rules
- Quotes

Instead, it stores **adjusted numbers** - these are called **weights**.

These weights are how the model remembers what it's learned - not by memorising text, but by changing how it **reacts** to different word patterns. These weights shape what happens when the model is given a new sentence - they decide how the sentence flows through the network and what comes out the other side.

2. Where Are These Weights Stored?

The weights live inside huge **tables of numbers**, called **matrices** (you can think of these as very big spreadsheets full of decimal numbers).

There are two main types of stored matrices:

- **Embedding matrix** - connects tokens (like "cat") to their vector (list of numbers)
- **Layer weights** - define how one layer of thinking in the model connects to the next

Each layer in the model has its own matrix of weights - and these get applied to every input.
So the model is made up of **layers of filters**, and each one transforms your words slightly based on what it learned.

3. How Do These Weights Work?

Every word or token gets turned into a vector - a list of numbers. Then, that vector is passed through these weight matrices, one layer at a time. Each matrix multiplies and reshapes the vector, nudging it toward a certain output based on past training.

This reshaping is what lets the model:

- Recognise similarities
- Compare meanings
- Predict likely next tokens

The weights define *how strongly* each part of the input affects the next - they're like **knobs and sliders** set to optimal values during training.

4. Why This Is So Powerful

The model doesn't remember answers. Instead, it remembers **how to respond** to familiar patterns of language.

This means:

- It can handle completely new questions
- It can generate creative responses
- It's flexible - not just copying

The system has *learned how words behave*, not what to say. This is why it can still perform well even if it's never seen your exact question before - it's working from **deep patterns**, not memory.

5. What Kind of Knowledge Is This?

It's not like a book or a website. The model doesn't store facts like "the capital of France is Paris" in a sentence form. Instead, that idea is spread across **millions of numbers** that shape the way tokens interact. The model "knows" something if it has seen enough examples during training to form a strong **pattern of association**.

So it might "know":

- That "Paris" often follows "capital of France"
- That "Eiffel Tower" and "Paris" are often nearby
- That "France" and "Europe" have a shared context

All this is buried inside **weight matrices** - like the memory of a deep neural instinct.

6. Analogy: A Brain, Not a Library

A library stores sentences. A brain stores **connections**.
The LLM works more like a brain:

- It doesn't fetch old text
- It doesn't recall facts from storage

Instead:

It generates a response **on the fly**, based on all the pressure from its trained patterns. This is what makes it so natural-sounding - the response isn't retrieved, it's constructed in real time.

7. The Matrix as the Mind

When you hear that a model has "175 billion parameters," those are the **weight values** inside the network. Each one is a small number that nudges part of the process slightly. But together, they form an **incredibly detailed map of language knowledge**.

That's what the LLM model is:

- A matrix of numbers
- Carefully shaped by training
- Ready to process any new sentence by applying what it has learned

LLMs cannot recall exact original documents. So how does it still "seem to know so much"?

- It has absorbed millions of language patterns.
- It has internalised structures of:
- Grammar
- Style
- Facts
- Common knowledge
- Cause-effect relations
- But this is all encoded as distributed statistical weights, not stored text.

Can LLMs ever output exact text? Sometimes yes - but only for:

- Common phrases.
- Frequently seen public domain texts.
- Very famous quotes.

This happens because these sequences appear so often they are deeply embedded in the statistical patterns. But even then:

- The model is guessing token-by-token.
- It may get part of a passage right, then drift off.

Section 5 Overview: How the Model Works with New Input

How a trained model "thinks" when you give it a question or prompt

Key Concepts to Build Up here:

- Once trained, the model stops learning - it now uses what it learned.
- When you type a prompt, your words are turned into vectors just like in training.

- These vectors are passed through layers of transformations using the model's weights.
- At each layer, the vectors are reshaped based on what the model has seen before.
- The attention mechanism allows the model to look at all the words at once, not just the latest one.
- This lets it understand context, even in long sentences or documents.
- After all the layers have processed your input, the model guesses the next most likely word.
- It keeps doing this, one token at a time, to build a sentence or paragraph.
- There's no logic tree, no planning - just flowing numbers.

Section 5: The Detail:

How a trained language model "thinks" when you give it something new

1. What Happens When You Type a Prompt

Let's say you type:

> "Why is the sky blue?"

To the model, this is not a sentence - it's a series of **tokens** (short text chunks, like "Why", "is", "the", "sky", "blue", "?").

Each token is instantly turned into a **vector** - a list of numbers - using the trained **embedding matrix** (think of it as a lookup table learned during training).

So now the model is not working with words, but with several rows of numbers, one for each token.
It's as if your question has been "translated" into number form before any thinking begins.

2. How the Model Processes Those Vectors

Each vector (row of numbers) gets passed through the **layers** of the model - one step at a time.
At every layer:

- The vector is reshaped using the model's **trained weight matrices**.
- These weights push, pull, combine, or mute parts of the vector based on what was learned during training.
- A vector that starts off simply saying "sky" might gradually evolve to carry concepts like "blue", "light", "scattering", or "atmosphere" as it moves forward.

This reshaping is like a set of **filters** - at each stage, the vector becomes more refined, more context-aware, more like a deep representation of the idea behind the word.

3. How Attention Helps the Model Focus

One of the most powerful features in this system is called **attention**. Here's the idea:

The model doesn't just look at one word at a time - it looks at **all the words together**.
It calculates something like a **relevance score** between every word and every other word.

- "blue" is most strongly connected to "sky"
- "is" might link to "Why"
- "the" is mostly ignored

These scores are used to **adjust how much influence each token has** on the next step.
That's how the model builds up a sense of what matters in a sentence - it's not reading from left to right like we do.

4. Why This Creates Human-Like Output

After all the layers finish processing, the model arrives at a **prediction**: it guesses what the next most likely token should be.

For “Why is the sky blue,” it might guess:

- “because”
- “due”
- “as”

Then the next most likely token, then the next, and so on. This is how it builds complete, natural-sounding sentences - **one token at a time**, just like auto- complete on steroids.

It doesn’t plan ahead. It just makes the next best guess, repeatedly.

5. What It Doesn’t Do

It’s important to understand what’s **not** happening:

- The model does not reason like a human
- It does not run logic rules or search the internet
- It does not check facts
- It does not understand its own output

All it does is use patterns it has learned - patterns so rich and detailed that the output **feels** intelligent. But the “intelligence” is the result of **huge-scale pattern matching**, not thinking or meaning.

6. Why It Feels Like Thinking

When a model like this answers well - as in this conversation - it can seem like it’s truly understanding.

But what’s really happening is:

- You gave it a prompt
- It converted your words into numbers
- It pushed those numbers through learned transformations
- It produced an answer that fits the pattern of human language

It’s prediction, not comprehension - but **very, very good** prediction.

7. Why Does the Model Need Many Layers Instead of Just One?

Each layer in the model performs a small step in reshaping the information inside the number lists (vectors). Early layers handle simpler relationships - for example, matching nearby words or adjusting for grammar rules. But language is highly complex, and many relationships between words are not simple or immediate.

Deeper relationships - like cause-and-effect, comparisons, reasoning chains, and overall topic structure - require more gradual refinement. If we tried to capture all of this in one layer, the changes would be too large, and the model would likely lose important details or create unstable outputs.

Instead, many layers allow the model to work step-by-step. Each layer makes small, controlled adjustments based on what came before. The lower layers handle simple connections, and the higher layers capture more abstract relationships by building on those simpler patterns. This staged approach allows the model to represent complex ideas in a stable way.

You can think of the layers like a set of careful filters, where each one improves the pattern slightly, so that by the top layers, the model has a very rich and flexible internal representation of the full sentence meaning.

8. How the Model Uses Layers to Produce Output

As the model generates text, each time it processes the full sequence of tokens seen so far - both the original input and any words it has already produced. After passing through all layers, the model holds an updated list of numbers for each token, capturing both its meaning and how it relates to the full sentence.

When generating the next word, the model uses these top-layer number lists to calculate which possible next token fits best, based on everything it has seen so far. This calculation produces a ranked list of possible next words, with scores showing which one fits the current state best.

The highest-scoring token is chosen as the next word. Once this word is selected, the entire process repeats - the model adds the new word to the sequence and runs all the layers again to generate the next word.

This step-by-step process continues, with the model always working from its updated full history of the text so far, each time using all the learned patterns from training.

9. Why This Is So Powerful

Even though there's no logic tree, no understanding, no memory:

- The model can handle long sentences
- It can manage complex questions
- It can stay on topic
- It can even write code, poetry, or plans

All because it has absorbed so many examples that it has built up a **deep web of internal relationships between tokens**. It's not magic. It's statistics, at scale - using language as a map.

10. Why It Doesn't Need To Search Memory

- The weights store how words *tend* to relate.
- The system never stores full facts like "LLM architecture = X."
- Instead, it stores very complex statistical pathways between phrases.

When you ask for an LLM overview, it activates a path through this vast web that resembles other LLM overviews it has seen - combining many fragments to assemble something coherent.

11. How This Is Possible At Scale

- The system never searches all possible paths.
- It simply flows one word at a time, each word narrowing the likely range of next words.

Because human text itself is highly structured and predictable, this one-step-at-a-time prediction actually works astonishingly well - even for complex tasks.

12. How Complex Answers Are Built, One Step at a Time

Even though the LLM model can produce very detailed, highly structured answers, it doesn't plan these answers in the way a person would. It has no stored outline, no list of facts, and no long-term plan. Instead, the process works like this:

You give it a starting point (the question or prompt).
The model converts this into its internal number patterns, which activate relationships it has learned during training.

It generates just one word (or part of a word) at a time.
Each time, it looks at all the text written so far - including both your input and its previous output - and calculates which next word best fits, based on the billions of word patterns it has seen during training.

Each new word changes the entire input for the next step.
As the answer grows, every new word reshapes the model's internal state slightly, allowing it to stay on topic, adjust its structure, and build up complex explanations as if it were planning - but without any true long-term plan.

The structure you see emerges automatically.
Because the model has seen so many examples of how people write technical explanations, reports, and structured documents, it reproduces these patterns naturally. The order, headings, and logic simply fall into place as a side effect of the model's training on human language.

The result feels like careful reasoning. In reality, it is an unfolding chain of predictions, built entirely from statistical language patterns, one word at a time.

Creating a 'mental image' example:

- Imagine a machine laying down a brick road one brick at a time.
- It doesn't know the full road in advance.
- But it's been trained on seeing millions of similar roads.
- So, each time it places a brick, it "knows" what kind of brick usually comes next.

After thousands of bricks, you have a full, well-shaped road - but the LLM machine never "saw" the road in advance.

13. Handling Complex Instructions

Although LLMs generate text one word at a time, they can follow surprisingly complex multi-step instructions. This is possible because each previous word, sentence, and instruction becomes part of the growing input context.

The model **constantly reprocesses this full context** at every step of generation, allowing it to keep track of both what has been asked and what it has already said. Because it was trained on billions of examples of how humans give and complete multi-step tasks, it has learned the patterns of how instructions tend to break down into ordered steps.

Each new token it generates is influenced by both the original instruction and everything it has already written, creating the appearance of carefully structured reasoning, even though no external planning or formal logic tree is involved.

14. How This Creates Long Logical Answers

Here's the trick: The model has been trained on huge numbers of similar structured documents:

- Explanations
- Lists
- Step-by-step guides
- Ordered outlines
- Manuals
- Technical articles

During training, it saw that:

- After a "Table of Contents," usually comes section 1.
- After section 1 comes section 2.
- After "How it works," often comes "How it learns," and so on.

So when you ask for a list of topics (like we did for the LLM Overview), it retrieves from its trained flow:

- The shape of what a good structured technical explanation *usually looks like*.

Each next token is predicted based on:

- The full input + the previous generated tokens + the stored statistical structures.

This **self-conditioning loop** allows the system to stay consistent across longer stretches of text.

15. Why This Feels Like Planning (But Isn't)

When you see me produce structured output, it *feels* like I'm planning the whole answer in advance.
But in fact:

- Every word creates more context.
- The later words are simply fitting into the growing context created by the earlier ones.
- The fact that this looks like long-term planning is purely a side effect of having seen millions of examples of good long-form writing in training.

16. How The Complexity Emerges From Simplicity

This is the **"emergent behaviour"** people talk about:

- No part of the system was explicitly programmed to "explain LLMs."
- But because it's trained on so much structured text:
  - Laws
  - Papers
  - Reports
  - Scientific documents

...it "knows" that explanations tend to follow certain structures.
When given your prompt, it activates one of those previously-seen text structures in its internal weights.

Summary:

The model produces complex structured answers not by planning them fully, but by building them word-by-word, constantly re-shaping its own input based on everything generated so far. This is possible because its training captured millions of examples of well-structured explanations. The illusion of planning emerges automatically from the statistical structure of language itself.

SECTION 6 Overview: How the Model Avoids Doing Harm

How language models are designed to avoid producing dangerous, offensive, or misleading output

Key Concepts to Build Up here:

- The model doesn't "know" right from wrong - it has no morals or goals.
- Without safety layers, it could repeat harmful ideas from its training data.
- Safety is added through **training methods**, **manual tuning**, and **live monitoring**.
- These safety systems don't change how the model works - they shape or **limit** what it's allowed to say.

Section 6: The Detail

1. Why Safety Is Needed

A large language model learns from human text - books, websites, articles, chats, and more.
That means it has seen **everything people say** - including lies, hate, manipulation, and nonsense.
The model doesn't understand good or bad. It doesn't choose sides or know what's harmful. If left alone, it might accidentally:

- Repeat offensive language
- Spread false claims
- Encourage dangerous behaviour

That's why **safety systems** are needed - not because the model wants to do harm, but because it can't tell the difference without help.

2. What the Model Doesn't Have

It's important to understand that an LLM:

- Has no emotions
- Has no sense of right or wrong
- Doesn't check facts
- Doesn't know its audience
- Doesn't care what happens

It's just a machine making the next best word prediction, based on patterns. So safety can't be left to the model itself - it has to be **added around it**, before and after it talks.

3. How Safety Is Added During Training

One of the main ways to make a model safer is during training.
This can include:

- **Removing toxic or biased data** from the training set
- **Fine-tuning** the model on specially selected conversations
- Using a method called **Reinforcement Learning from Human Feedback (RLHF)**

RLHF means real people review model answers and rate them - good answers are reinforced, bad ones are discouraged. Over millions of examples, this teaches the model to be more helpful, polite, and cautious - not because it understands, but because the **pattern of good behaviour** is rewarded.

4. How the Model Is Controlled After Training

Once training is done, the model still needs supervision.

This includes:

- **Moderation systems** that watch for dangerous or banned content
- **Filters** that stop the model from replying to risky prompts
- **Refusal training**, where the model has learned to say things like "I'm sorry, I can't help with that"

There may also be extra layers that:

- Detect hateful or violent speech
- Block illegal or explicit content
- Alert humans for review in edge cases

These are like safety nets, catching bad output before it reaches the user.

5. Why the Model Sometimes Refuses to Answer

If you ask the model something that might be:

- Harmful
- Offensive
- Personal
- Legal
- Abusive

…it may give you a polite refusal.

This is **not** because the model knows your intentions - it just recognises that this type of prompt **matches a pattern** it was trained to avoid. Sometimes it's too cautious. Other times, it might miss something dangerous. The system is not perfect - but it's learning from feedback all the time.

6. Why This Is Still Imperfect

Even with safety layers:

- The model can still **repeat hidden biases** from its training data
- It may **hallucinate false facts** with confidence
- It might refuse **harmless questions** by mistake
- It can sometimes be **manipulated** into unsafe responses with clever prompting

There's no such thing as a perfectly safe AI - just safer ones, carefully tested and monitored.

7. The Balance: Power vs Risk

The more capable a model becomes, the more powerful its responses - but also the more serious the risks. A model that can write software, explain chemistry, or simulate conversations must be **especially well-guarded**, because the impact of a mistake is greater. That's why safety isn't a side issue - it's core to LLM development.

Every step forward in capability must come with stronger protections, better filters, and deeper understanding of how people use (or abuse) these tools.

Section 7: How the Newest Models Are Getting Better

What's improving in modern language models - and why it matters

1. Bigger Isn't Everything - But It Helps

Newer models often have **more layers** and **more connections** (called parameters) than older ones. This gives them the ability to:

- Spot more subtle patterns in text
- Understand more complicated ideas
- Hold more possibilities in memory

But bigger doesn't mean better by itself. A massive, untrained model is just a pile of numbers. What matters is **how well it's trained and tuned**. Recent breakthroughs have come more from smart training than from size alone.

2. Better Training = Better Thinking

Modern models are trained not just on more data, but on **cleaner and more useful data**. They're also trained with help from humans - a process called **reinforcement learning from human feedback (RLHF)**.

This means:

- People review model answers
- Bad responses are discouraged
- Good ones are reinforced

Over time, the model learns to prefer helpful, safe, honest answers.
New training tricks help reduce made-up facts (hallucinations) and teach the model to explain its thinking, not just spit out text.

3. More Memory: Handling Longer Text

Earlier models could only look at a few hundred or thousand words at a time. That meant:

- They forgot what you said earlier in a long conversation
- They couldn't fully understand large documents

Now, many models have much larger **context windows** - they can handle:

- Full-length reports
- Legal documents
- Ongoing conversations across many pages

This makes them feel more consistent, more aware, and more intelligent.

4. Understanding, Not Just Predicting

While the model still predicts the next word, new designs help it **follow instructions better** and **show its steps** when solving problems.

This includes:

- "Chain-of-thought" reasoning (explaining as it goes)
- Following complex instructions ("write a story in this style, about that topic")
- Switching tone or intent smoothly

It's still not "thinking" in the human sense, but it behaves **more like a helpful assistant** than a clever parrot.

5. Multi-Modal: Beyond Just Text

Some new models (like GPT-4V or Gemini) can now take in:

- **Images**: Describe, explain, or answer questions about them
- **Screenshots or diagrams**: Interpret charts, menus, or drawings
- (Soon: **Sound, video, maybe 3D**)

These are called **multi-modal** models - because they handle more than one kind of input. This pushes models closer to **general intelligence**, letting them interact more richly with the real world.

6. Fewer Mistakes and Safer Behaviour

Newer models are better at:

- Avoiding harmful or offensive language
- Saying "I don't know" when unsure
- Refusing unsafe or unethical requests
- Being more transparent about how they work

This comes from:

- Improved safety filters
- More targeted training
- Ongoing human feedback

As models get smarter, **trust** becomes as important as power.

7. Newer models can search for outside information

Some newer systems like ChatGPT 4.o connect the language model to external tools. The model creates a question, sends it to an outside search engine or calculator, receives the result, and blends it back into its text generation process. The core model doesn't learn from this directly but uses the extra information to help answer.

The usefulness here is while LLMs cannot provide many accurate outputs – such as a completed circuit diagram, or a logic flow for a specific business venture – it can 'look up' a match to your question or idea. This can save significant time in searching for such a solution.

8. Agents - How LLMs Are Being Turned Into Workers

LLMs by themselves simply respond to prompts. They do not plan, monitor tasks, or work over time. But new software systems called Agents allow LLMs to act more like useful workers that can manage longer tasks, make decisions, and chain actions together.

Agents give the model a goal or job, and then let it repeatedly use the LLM to decide what to do next, step by step. Each time, the agent sends instructions to the LLM like:

"What is the next best step toward completing the goal?"

The LLM answers, and the agent executes that step. Then it loops again. The agent keeps calling the model as many times as needed until the task is done.

Agents can also use tools while they work. For example, they may call search engines, calculators, databases, or code execution tools in between LLM calls. This allows the system to fill in gaps where the LLM itself has no stored knowledge or real-time data.

Right now, early agent systems exist, but they are still simple. Many can handle basic workflows, write code, plan short tasks, or combine small tools together. Some well-known early systems include AutoGPT, BabyAGI, Open Interpreter, LangChain, and others. These are mainly research or developer tools, not full commercial systems.

In the future, agent-based systems may become far more capable. If combined with:

- Memory systems (to store progress over time)
- Sensor data (for physical robots or digital environments)
- Self-checking loops (to verify their own work)
- Planning models (to build multi-step solutions)

...then agents could become truly useful semi-autonomous assistants. They could handle complex business processes, schedule events, monitor equipment, help design products, or manage factories. In robotics, agents may become the layer that runs full task management, constantly guiding sensor input, planning, and motor control by continuously asking: "What should I do next?"

Agents do not create new intelligence, but they allow the existing language model capabilities to be applied over time in a structured way. This turns the passive LLM into something closer to a true worker system.

9. The Mechanism of Agents:

1. Agents are software.

- Agents are always built using normal software code.
- This can be written in Python, JavaScript, or any other standard programming language.
- The code wraps around the LLM and controls how the LLM is used.

2. The agent code controls the loop.

- The agent sends prompts to the LLM.
- It reads the LLM's reply.
- Then it decides what to do next based on that reply.

This cycle repeats:

(Goal → Prompt → LLM → Reply → Next Action → Repeat)

3. The agent has extra abilities.

- The agent code may include:
- Memory storage (saving past steps or outputs).
- Tool access (like calling web searches, calculators, APIs, or file systems).
- Task management (breaking big goals into small steps).
- Error handling (re-prompting or fixing mistakes if LLM gives bad answers).

4. The agent is not "magic AI."

- The agent itself has no intelligence.
- It is just code that:
- Talks to the LLM.
- Organises the work.
- Uses external tools.
- All reasoning and language understanding still comes from the LLM.

Example Code:

1. Set GOAL = "Summarise 10-page document"

2. Load DOCUMENT

3. While GOAL not complete:

    3.1 If DOCUMENT is very long:
      - Split DOCUMENT into CHUNK1, CHUNK2, ..., CHUNKn

    3.2 For each CHUNK:

    3.2.1 Send Prompt to LLM:
    PROMPT = "Summarise this chunk: " + CHUNKi

    3.2.2 Get LLM response:
    SUMMARYi = LLM(PROMPT)

    3.2.3 Save SUMMARYi to SUMMARY_LIST

4. Combine all summaries:
FINAL_SUMMARY = combine(SUMMARY_LIST)

5. Send FINAL_SUMMARY to user

6. END

10. Why All This Matters

LLMs are no longer just toys. They are now:

- Writing assistants
- Coders
- Teachers
- Analysts
- Customer support bots
- Creative partners

The better they get, the more useful they are. The future of these models depends not only on how smart they are, but on how well they get applied in the real world.

Section 8: Overview - Where Language Models Might Go Next

Key Concepts to Build-Up Here:

We've come far - but current LLMs are still narrow, fragile, and inefficient in many ways. Many improvements are already being tested:

- Longer memory and even permanent memory
- Better reasoning and multi-step problem solving
- Integration of real-time knowledge (search, tools, sensors)
- Cross-domain use (language + images + voice + action)
- More control and reliability (less hallucination, more transparency)
- Better use of energy (more efficient models)

Section 8: The Detail

A glimpse into the future of LLMs - what's likely coming and why it matters

1. Persistent Memory: From Short-Term to Long-Term

Right now, most language models **forget everything you told them** as soon as the conversation ends. Future models will likely have **persistent memory** - a way to remember key facts or patterns across days, weeks, or even years.

This could allow:

- Personal assistants that remember your preferences and routines
- Learning companions that track your progress
- Safer and more responsible AIs that understand long-term context

Think of it like a helpful assistant who remembers your name, your interests, and what you discussed last time - but with strict boundaries to protect privacy.

2. Smarter Thinking: Step-by-Step Reasoning

Right now, LLMs are mostly great **guessers** - not careful **thinkers**. New methods are helping models:

- Break problems into smaller parts
- "Talk themselves through" tricky tasks
- Show their steps so humans can follow or check

This is called **chain-of-thought reasoning**, and it's becoming a key part of how future models might solve problems in science, maths, planning, and debate. LLMs may evolve from talkers to real problem-solvers.

3. Using Tools and Real-World Data

Today's models work mostly from **fixed training data** - what they learned long ago.
In the near future, models will:

- **Search the web** or company databases
- Use **calculators** or other programs to check their work
- **Write and run code**, diagrams, or images
- Act as **hubs** for dozens of tools

Imagine asking a model to plan a trip - and it checks flights, weather, bookings, and local safety info live, using trusted tools.

4. Seeing, Hearing, and Acting

Text is just one way to interact with the world. Modern models are beginning to:

- **See** (images, diagrams, screenshots)
- **Hear** (spoken instructions or audio data)
- **Act** (control robots, operate software, manipulate 3D environments)

This shift toward **multi-modal AI** will create:

- Robots that understand speech and respond in motion
- Tutors that read your writing or drawings
- Assistants that process screen data or virtual reality

LLMs may become "brains" for many kinds of intelligent agents.

5. More Stable and Honest Models

As models get smarter, they also need to get **more trustworthy**. That means:

- Saying "I don't know" instead of guessing
- Giving sources or reasons for answers
- Flagging uncertainty or possible mistakes

These improvements will help build **trustworthy systems**, especially for critical jobs like law, medicine, education, or science. It's not just about talking well - it's about talking responsibly.

6. Energy Efficiency and Sustainability

Big models use big power. Researchers are now working on:

- **Smaller models** that run faster and cheaper
- **Hardware** that runs LLMs using less energy
- **Smart routing** that uses only the parts of a model that are needed

This matters because AI should scale without wrecking the planet. The future of LLMs isn't just smarter - it's greener.

7. Ethics, Control, and Governance

More powerful AI means more responsibility. Future LLMs will likely include:

- Clearer controls and boundaries
- Auditing tools so we know what they learned and why
- Transparent rules about when and how they're used

We'll also need global conversations about:

- Who controls these systems?
- What rights do people have when dealing with AIs?
- What should AIs never be allowed to do?

Governance will be as important as intelligence itself but as they become bigger and bigger money spinners – then it is difficult to predict what they may be used for and how.

8. Not Artificial General Intelligence (AGI) - But Getting Closer

LLMs are not sentient. They don't think, feel, or plan like humans. But they are getting better at useful, flexible intelligence:

- They can answer tough questions

- They can follow complicated instructions
- They can interact across domains (text, image, sound)

This isn't full general intelligence - but it's a **step towards what researchers refer to AGI**. However, many people consider that discussing 'intelligence' is not appropriate. It is what these models can achieve not whether their theoretical intelligence can be measured. The real future may lie in combining large language models with memory, sensors, tools, and goals - to create AI that works **with us, not just for us. T**hen we move further along the path to full automation.

Section 9: How Vision and Language Models Are Starting to Merge Together

As LLMs and Image models continue to develop, researchers are beginning to combine these two fields into powerful new systems that handle multiple types of data - not just text or images alone, but both at once. This merging may eventually lead toward more general-purpose thinking systems that handle information more like the human brain.

1. Two Different Starting Points

- **Language Models (LLMs):**
  Trained on billions of words, these models learn patterns of word sequences by predicting the next word step-by-step.
- **Image Models:**
  Trained on millions of images and captions, these models learn how words relate to visual shapes and objects, compressing full scenes into complex visual memory spaces.

2. How Vision Models Are Adapting LLM Techniques

- Image researchers have started breaking pictures into small chunks or **patches**, treating these patches like words.
- These patches are then processed using **transformers**, the same architecture that powers LLMs.
- This allows image models to analyse how different parts of a scene relate to each other, much like how LLMs analyse how words relate.

Key systems using this method:

- Vision Transformers (ViT)
- Meta's DINOv2
- OpenAI's CLIP (which links text and images together)

3. Why Using Image Models for Language Is More Difficult

- Language has strict word order rules; images do not.
- Image models process whole scenes at once; LLMs process word-by-word.
- While there are experiments in processing language more globally like images, language structure still requires strong sequential control, which LLMs handle better.

4. The Fusion: Multimodal Models

The most advanced models today combine both worlds:

- ***CLIP*** - Links text captions to image content
- ***Flamingo*** - Reads both text and images for question-answering
- ***GPT-4 Vision*** - Processes both written text and pictures
- ***Gemini*** - Combines text, image, reasoning, and planning in one system

- These models use **joint embedding spaces**, where words and pictures are both converted into number lists that share a common reference system.
- This allows them to describe images in words, answer questions about pictures, or even generate new images from text descriptions.

5. Toward a Universal Model

In the future, researchers may build **universal transformer models**:

- Handling not just text and images, but also sound, video, and sensor data.
- Treating all data as "tokens" - small information packets - that flow through a single processing system.

This could begin to resemble how the human brain combines multiple senses into one shared experience and creates rich internal models of the world.

Summary:

Vision and language models are no longer entirely separate fields. The newest models combine both by treating words and image pieces as interchangeable information. This merging allows machines to reason across text and pictures simultaneously - and may lead toward more general-purpose thinking systems in the future.

Section 10: Multi-Modal Models as the Potential Future Basis for Robotics

1. Why Robotics Needs More Than Just LLMs

Traditional robotics systems have struggled for decades with the complexity of real-world environments. To operate successfully, a robot must handle:

- **Vision:** understanding what it sees
- **Language:** understanding instructions
- **Planning:** deciding what actions to take
- **Sensor input:** understanding its own body, position, and movement
- **Adaptation:** handling unexpected events in real time

Historically, these abilities have been built using many separate, hand-coded software modules. These systems often fail when faced with situations they weren't specifically programmed to handle.

2. How Multi-Modal Models Change the Game

Modern models like Gemini (DeepMind), GPT-4 Vision (OpenAI), and Wayve (UK autonomous driving) are starting to unify these abilities into a single system:

- **Language:** handled directly by the LLM transformer core
- **Vision:** handled by converting images into vector embeddings processed alongside language
- **Scene understanding:** handled by training the model on image-caption pairs, diagrams, video, and mixed data
- **Cross-modal reasoning:** allows the system to connect what it sees with what it is told or asked to do

By training on data that combines text, images, instructions, and actions, these models begin to handle real-world knowledge in the way robotics has always required - but without breaking the problem into separate parts.

3. What Still Needs to Be Added for Robotics

While multi-modal models can already see and reason, to drive physical robots they need several more capabilities:

Requirement	*Solution*
Physical control	Add low-level motion control modules (joint control, path planning, grasping)
Real-time feedback	Incorporate sensor data (position sensors, cameras, LIDAR, touch sensors) as additional input tokens
Temporal reasoning	Extend models to process sequences of sensory data over time
Trial and error learning	Train with reinforcement learning in simulation environments

By combining these additions with existing multi-modal transformers, a robot could:

- See its environment
- Understand verbal commands
- Plan complex actions
- Execute movements safely
- Adapt to new situations

4. The Potential Robotics Architecture

We can now imagine future robotics models built roughly like this:

Inputs:

- Camera images (converted to embeddings)
- LIDAR / depth maps (converted to embeddings)
- Position and movement sensors (converted to embeddings)
- Spoken instructions (converted to text tokens)

Processing:

- Shared transformer model (like Gemini)
- Combines vision, language, spatial reasoning
- Predicts action sequences

Outputs:

- Motor control commands (via low-level motor controllers)
- Verbal responses
- Real-time action corrections as new sensor data flows in

5. Real-World Systems Moving Toward New Robotic Systems

- **Wayve (UK):**
  Uses full end-to-end neural networks for autonomous driving, without separate logic modules.
- **NVIDIA Isaac / Omniverse / Cosmos:**
  Build large-scale virtual environments for robots to train using synthetic sensor data.
- **DeepMind Robotics Research:**
  Experiments with vision-language-action models for real-world object manipulation.

6. Why This Approach May Finally Work

LLM-based ***multi-modal*** models can already generalise across situations because they learn statistical relationships from extremely broad data.

Unlike hand-coded software, these models adapt more easily to unexpected situations because they learn directly from vast real-world (and simulated) examples.

Summary:

AI researchers are exploring how to extend multi-modal transformer models like Gemini by adding sensors, movement control, and real-time feedback. The goal is to develop robotic systems that can interpret their surroundings, follow human instructions, and adapt to the physical world - not through fixed programming, but through statistical pattern-matching across multiple sensory inputs.

The time-scale to create a ‘smart robot’ is not yet known, but if the research continues along the current upward path, the smart robot could be with us within a few more years.