Tab 1

Large Language Models Fundamentals

Introductory Workshop on LLMs and Prompt Engineering. Large Language Models in Digital Humanities Research. Summer School. Cologne. 8-11. September

- Lecture manuscript
- Christopher Pollin, supported by Claude Opus 4.1
- Slides: LLM-DH-School-Introduction-25

How LLMs Work

Large Language Models (LLMs) are optimised to **predict the next token**. They calculate the **probability distribution for the next token in a given sequence**, apply **temperature**-based sampling to control randomness, and select one token from this distribution. LLMs are **autoregressive**, meaning that each generated token becomes part of the context for predicting the next token. This creates a cascading effect where early choices shape all subsequent generations. While the generation process is sequential, the models use parallel attention mechanisms to process relationships between all tokens in their **context window** simultaneously. This fixed-size memory buffer limits the amount of prior text that can be considered. The result is like a butterfly effect: selecting “The king” versus “The peasant” as the opening words leads to different narratives, with each choice constraining what can plausibly follow.[1]

Scaling

Increasing the number of parameters (the adjustable weights that determine how the model processes information) and the amount of training data improves these models' ability to predict tokens. This follows empirical scaling relationships (scaling laws, which are not laws) that demonstrate consistent performance enhancements as the size of the model, the size of the dataset, and the computing power increase in tandem. However, these improvements follow an inverse power law, meaning that each advance requires exponentially more resources.

The GPT-2 model with 1.5 billion parameters could produce coherent paragraphs, but it often lost track of longer narratives. GPT-3, with 175 billion parameters, could produce sustained narratives and demonstrate few-shot learning. GPT-4, estimated to have over 1 trillion parameters, can handle 'reasoning', produce working code and follow multi-step instructions. GPT-5, released in 2025, is slightly better at all capabilities, reduces hallucinations, and is cheaper to infer.

---

[1] For an accessible introduction to some of the concepts, see Ethan Mollick. Thinking Like an AI (2024). https://www.oneusefulthing.org/p/thinking-like-an-ai. Andrej Karpathy. [1hr Talk] Intro to Large Language Models. https://youtu.be/zjkBMFhNj_g

Whether traditional scaling has reached a plateau remains contentious. The curve shows diminishing returns are inherent to the inverse power law, not necessarily a hard limit. However, practical bottlenecks are emerging. Data is finite since we have only one internet to scrape. Compute requirements are growing exponentially, demanding massive GPU clusters. Energy consumption for training runs now rivals cities.

These constraints push researchers toward alternative scaling paradigms. Test-time compute scaling allows models to "think longer" on difficult problems, as demonstrated by reasoning models that improve performance by generating longer chains of thought. Multi-agent systems achieve complexity through orchestration rather than single-model scaling, breaking tasks into components that simpler models can handle. Synthetic data generation attempts to bypass some of the data bottleneck.

Ilya Sutskever's statement that "pretraining as we know it will end" does not reflect a failure of scaling laws, but rather a shift in what we are scaling. The question is no longer whether scaling will continue, but which dimensions we will scale next. When I write "we", I am not referring to everyone. It is the big tech labs — and only the big tech labs.

The Three Eras of LLM Training

But it is not only **pre-training** (compressed knowledge from vast internet text) that forms the models, but also **post-training**. While supervised fine-tuning taught models to follow instructions through Q&A examples created by contract workers, Karpathy argues this second era is "going away". Pre-training is like compressed knowledge and patterns[2], while post-training techniques (RLHF[3], instruction tuning[4], constitutional AI[5], etc.) teach how to use those patterns and knowledge effectively. Andrej Karpathy describes this evolution as three distinct eras: pre-training on internet text, supervised fine-tuning on conversations, and an emerging third **era of reinforcement learning (RF) through environments** (synthetic data) where models actually interact, take actions, and see outcomes. This RL approach remains experimental and reward functions are problematic ("*super sus*"[6] in Karpathy's words) and haven't been properly invented and scaled yet. For this emerging paradigm, Genie 3[7] demonstrates where this might lead: creating virtual worlds where

---

[2] Think of these patterns as Chollet's 'programmes'.
[3] Reinforcement Learning from Human Feedback (RLHF) Explained. https://youtu.be/T_X4XFwKX8k
[4] Generative AI for Everyone. DeepLearning.AI. https://www.coursera.org/learn/generative-ai-for-everyone/lecture/oxPGS/how-llms-follow-instructions-instruction-tuning-and-rlhf-optional
[5] https://www.anthropic.com/research/constitutional-ai-harmlessness-from-ai-feedback. Bai, Yuntao, Saurav Kadavath, Sandipan Kundu, et al. 'Constitutional AI: Harmlessness from AI Feedback'. arXiv:2212.08073. Preprint, arXiv, 15 December 2022. https://doi.org/10.48550/arXiv.2212.08073.
[6] "Super sus" is internet slang for "super suspicious" or "highly questionable". I had to look it up too.
[7] Genie 3 (Google DeepMind, 2025) is an interactive foundation world model that generates playable 3D environments from single images in real-time. Unlike its predecessors, it allows continuous agent interaction while maintaining visual consistency, representing a shift from passive video generation to active world simulation. https://deepmind.google/discover/blog/genie-3-a-new-frontier-for-world-models

agents can interact and maybe learn from their environments. While current LLMs learn from static text, future models may learn in new ways, through interaction, experimentation, and simulated embodied experience[8] in virtual worlds.

LLMs as ‘Retrieval-ish’ Systems or/and ‘Program’ Retrieval

At the fundamental level, LLMs perform next-token prediction, but one theoretical interpretation suggests this process works by retrieving and executing learned computational patterns (*‘vector programs’*) from latent space. According to François Chollet[9], LLMs function as pattern retrieval systems. When you query an LLM, your input acts as coordinates in a vast latent space containing millions of *‘vector programs’* learned from internet text, but increasingly synthetic multimodal data. These *‘vector programs’* aren't traditional code but learned transformations that map inputs to outputs, making the model essentially a retrieval system for computational patterns. These *‘programs’* encode both declarative knowledge about facts and procedural knowledge about how to perform tasks, compressed into high-dimensional mathematical structures. The model retrieves the nearest matching program and executes it on your input, with the output bearing traces of the specific training data that formed that pattern. It should be noted that this is just one of several possible explanations for what might happen in an LLM.

This offers one explanation for the patchy generalization we observe. Models excel at implementing a snake game in Python, where thousands of training examples exist, yet struggle with the same task in obscure programming languages where examples are sparse. Nevertheless, as models scale and training improves, they demonstrate increasing capability in code generation as a general task, transferring patterns learned from common languages to less common ones more effectively. Chollet originally argued this demonstrates the absence of true intelligence, specifically the inability to adapt to genuinely novel situations.[10] However, following o3-preview's achievement of 88.5% on the ARC-AGI benchmark (with high compute), Chollet has revised his position, acknowledging that while extremely inefficient, this represents genuine program synthesis with some ability to handle novel problems.[11]

Models can interpolate between adjacent programs, blending characteristics to create apparently novel combinations. A request for “Shakespearean explanation of quantum computing” would merge patterns from both domains. However, models cannot extrapolate beyond the the learned probability distribution underlying their training data[12]or synthesize genuinely new programs that fall outside this distribution. When no appropriate pattern exists, they retrieve the nearest available one, regardless of its actual relevance. This can

---

[8] Generalist Embodied Agent Research. https://research.nvidia.com/labs/gear
[9] François Chollet - Creating Keras 3. https://youtu.be/JDaMpwCiiJU
[10] Chollet, François. ‘On the Measure of Intelligence’. arXiv:1911.01547. Preprint, arXiv, 25 November 2019. https://doi.org/10.48550/arXiv.1911.01547.
[11] https://arcprize.org/blog/oai-o3-pub-breakthrough
[12] See slide showing how “New Discoveries” lie outside the boundary of current human knowledge

result in fluent but incorrect outputs, as there are no mechanisms in place to signal retrieval failure. In other words, LLMs *'hallucinate'* or, more accurately, *'confabulate'* the answer.

This perspective might explain why prompt engineering is not trivial. Minor phrasing changes shift coordinates in program space, potentially crossing into different regions. "Explain quantum mechanics simply" might retrieve educational content patterns while "ELI5[13] quantum mechanics" could activate Reddit community explanation patterns. In this view, the model cannot evaluate which program is more appropriate, only which is nearest in vector space. It finds, executes, and combines programs into outputs that may appear to demonstrate understanding. Prompts like "*let's think step by step*" or other *Chain of Thought* techniques can also be understood as retrieving specific '*reasoning programs*' learned from training data rather than invoking genuine metacognitive processes. Models do not think; they execute patterns that mimic thinking. As these patterns become more sophisticated, their functional output increasingly resembles genuine reasoning, despite the fact that the underlying mechanism remains fundamentally different.[14]

What counts as "genuinely novel" remains debated. Is combining existing patterns into new configurations true creativity? When models use Python interpreters and web searches, are they transcending retrieval, or are they simply following learned patterns of tool use? What if multiple agentic systems work together on a task to simulate a project and achieve a goal? Other researchers propose different interpretations. Ilya Sutskever[15] argues that sufficiently accurate next-token prediction necessarily develops world models and causal understanding. Some see emergent reasoning capabilities beyond pattern matching. The retrieval framework provides insights for understanding certain limitations and prompt engineering strategies, though it may not fully explain all observed model behaviors. If this architectural interpretation holds, it would suggest that scaling and training alone may not transform these systems beyond sophisticated retrieval mechanisms, remaining constrained by patterns encoded during training. By September 2025, it seems that LLMs alone will no longer be enough. However, LLMs are no longer alone; they have become agentic systems within information systems.

Pre-Training ("Compression of Knowledge")[16]

The pre-training phase of Large Language Models transforms vast quantities of text data into model parameters through next-token prediction. For multimodal models, this extends to images, audio, and video through joint transformer architectures that process all

---

[13] ELI5 stands for "Explain Like I'm 5", a Reddit-originated phrase requesting simple explanations of complex topics.
[14] Summerfield, Christopher. These Strange New Minds: How AI Learned to Talk and What It Means. Viking, 2025.
[15] Ilya Sutskever (OpenAI Chief Scientist) — Why next-token prediction could surpass human intelligence. https://youtu.be/Yf1o0TQzry8
[16] Andrej Karpathy. How I use LLMs. https://youtu.be/EWvNQjAaOHw. Andrej Karpathy. [1hr Talk] Intro to Large Language Models. https://www.youtube.com/zjkBMFhNj_g

modalities simultaneously in a unified semantic space using masked training.[17] The training ingests trillions of tokens from web sources and synthetic datasets, teaching the model to predict what comes next in a sequence. The model learns by repeatedly guessing the next token based on previous ones, then adjusting its parameters to improve these predictions. This mathematical process captures statistical patterns in language.

The resulting model exhibits specific operational characteristics. The compression is lossy, meaning the model cannot perfectly recall[18] its training data; instead, it encodes statistical patterns and relationships. This lossy nature strikes a balance between generalisation and memorisation, enabling the model to handle novel inputs to some extent. The encoded knowledge is probabilistic and is represented as learned distributions rather than discrete facts. Training establishes a temporal boundary, meaning that the model's knowledge remains fixed at the point when training concludes.

Pre-training demands substantial resources! Infrastructure investment reaches hundreds of billions of dollars, while the process consumes months of continuous computation on specialized GPU clusters and significant energy. Data curation, researcher time, and the opportunity cost of compute resources add to the total investment. These requirements constrain who can develop (powerful) foundation models. In this context, a foundation model is the raw, pre-trained neural network that results from the resource-intensive pre-training process, designed to be adapted for multiple downstream tasks rather than a single application.

Through this process, the model learns multiple layers of structure: syntactic patterns, semantic relationships, some aspects of *'world knowledge'*[19], and *'reasoning'* procedures. At scale, models exhibit (weak) emergent capabilities not explicitly programmed, including multi-step reasoning, cross-lingual transfer, and in-context learning.

Pre-trained LLM therefore function as **lossy, probabilistic compressions of their training data, encoding trillions of tokens into billions of parameters**. This compression trades perfect recall for pattern recognition and generalization, producing models that generate coherent outputs by sampling from learned distributions. While the compression metaphor illuminates storage efficiency and information density, pre-training simultaneously

---

[17] Cheng, Ho Kei, Masato Ishii, Akio Hayakawa, Takashi Shibuya, Alexander Schwing, and Yuki Mitsufuji. 'Taming Multimodal Joint Training for High-Quality Video-to-Audio Synthesis'. arXiv:2412.15322. Version 1. Preprint, arXiv, 19 December 2024. https://doi.org/10.48550/arXiv.2412.15322.

[18] While models generally cannot perfectly recall training data, exceptions exist where verbatim memorization occurs, particularly for sequences that appear multiple times in training data. Carlini, Nicholas, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. 'Quantifying Memorization Across Neural Language Models'. arXiv:2202.07646. Preprint, arXiv, 6 March 2023. https://doi.org/10.48550/arXiv.2202.07646

[19] It only learns about the world through text, images, and videos. As Yann LeCunn would say: "*Every cat is smarter than an LLM*".

accomplishes representation learning and capability development that extend beyond simple data compression.

The “Gestalt” of a Zebra Wikipedia Article

LLM cannot access Wikipedia articles directly. They have what Karpathy calls a "Gestalt" of the text comprising compressed statistical patterns learned during training. When generating text about zebras, models use probabilistic next-token prediction based on these patterns, not retrieval of stored documents. This explains why models can discuss zebra species and habitats coherently but cannot reproduce exact Wikipedia phrasing like "Die Zebras (Hippotigris) sind eine Untergattung" or count article characters. Models work from parametric knowledge embedded in weights. They access external sources only through explicit tool use such as web search functions. This distinction between internal pattern-based generation and tool-mediated retrieval defines how LLMs process information.

The USA is investing Hundreds of Billions in Data Centres and Energy Production.

Modern LLMs have achieved significant efficiency gains, with energy consumption per prompt dropping to just 0.0003 kWh in 2025, equivalent to 8-10 seconds of Netflix streaming, representing a 33x improvement in just one year according to Google. However, water usage remains less certain, ranging from a few drops to approximately 5 ml per prompt, depending on the measurement methodology used. These dramatic efficiency improvements, coupled with plummeting costs (from $50 per million tokens for GPT-4 to just 14 cents for the more capable GPT-5 Nano), have made powerful AI economically viable for over a billion users worldwide. While these runtime figures exclude the substantial one-time energy investment required for training (GPT-4 required an estimated 500,000 kWh), the marginal environmental impact per query has become negligible. This has enabled the era of 'Mass Intelligence', in which advanced AI is as accessible as a Google search, as described in the article. This is from a post from Ethan Mollick.[20] The critical question is whether the efficiency gains of 33x can outpace growth in usage. Nevertheless, it uses gigantic amounts of resources and energy!

While individual AI queries appear efficient (GPT-4o uses 0.43 Wh per query), scaling to 700 million daily queries creates massive environmental impacts equivalent to powering 35,000 homes, consuming water for 1.2 million people, and emitting carbon requiring a Chicago-sized forest to offset. Infrastructure choices matter more than model size (GPT-4o mini uses 20% more energy than GPT-4o despite being smaller due to older hardware), and as AI becomes cheaper and more accessible, total resource consumption is exploding

[20] Ethan Mollick. Mass Intelligence. From GPT-5 to nano banana: everyone is getting access to powerful AI https://www.oneusefulthing.org/p/mass-intelligence

faster than efficiency gains, with inference now accounting for 90% of lifetime energy use, potentially costing 1,400x more annually than initial training for applications like Google Search.[21]

Tokenization

Tokenization transforms text into numerical units for LLM processing. The tokenization strategy prioritizes computational efficiency by minimizing sequence length The process transforms raw text through extraction, cleaning, segmentation into tokens, and mapping to numerical IDs. For example, “`Hello World!`” becomes three tokens `['Hello', 'World', '!']` mapped to `[13225, 5922, 0]`.

Modern LLMs use **subword tokenization** to balance the tradeoff between character-level approaches (small vocabulary but inefficiently long sequences for quadratic attention) and word-level approaches (semantically rich but massive vocabularies with out-of-vocabulary problems). Subword methods create tokens ranging from characters to complete words based on training frequency.

Byte-Pair Encoding (BPE) builds token vocabularies through statistical compression. Beginning with 256 UTF-8 bytes, BPE scans training data for the most frequent adjacent token pairs—like `‘h’+’u’` appearing thousands of times—and merges them into new tokens (`‘hu’`→`256`). Through 50,000-100,000 such merges, common sequences compress dramatically: "the" becomes token 264, while rare words like "bioluminescent" decompose into known subwords `[bio][lumin][escent]`, eliminating vocabulary gaps.

Models struggle with character-level tasks like spelling or letter counting since they process subword units. Improved whitespace handling between GPT-2 and GPT-4, particularly merging multiple spaces, substantially enhanced coding capabilities.

Token count determines API costs and context utilization, with standard approximations of 100 tokens per 75 English words (1.3 tokens/word) and 4:1 character-to-token ratios varying by language and content. Critical failures arise from training misalignment. Tokens frequent in tokenizer data but absent from model training retain untrained embeddings, causing unpredictable outputs. Special tokens like `<|endoftext|>` mark document boundaries but require careful handling to distinguish literal text from structural commands.

Hands-On: Try Tokenization Yourself!

Spacing changes token counts: 'Hello' uses 1 token while 'H e l l o' requires 5-10 tokens as each character and space tokenizes separately. Languages tokenize differently, with Chinese characters often representing whole words per token while agglutinative languages

[21] Jegham, Nidhal, Marwen Abdelatti, Lassad Elmoubarki, and Abdeltawab Hendawi. ‘How Hungry Is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference’. 14 May 2025. https://doi.org/10.48550/arXiv.2505.09598.

need multiple tokens per word. Typos fragment into unexpected subword patterns, degrading comprehension. Format selection impacts costs as verbose XML tags consume more tokens than equivalent JSON. Understanding tokenisation allows for quick optimisation, cost reduction and efficient use of context. However, it reveals which model you are using!

Why do you see so many em dashes and colons now?[22]

Em dash frequency in AI-generated text increased sharply between model generations. GPT-3.5 produced zero em dashes in controlled tests, GPT-4.0 produced 14, and GPT-4o produced 16 under identical conditions. This proliferation stems from tokenization economics: an em dash with spaces occupies one token in the cl100k_base tokenizer while ", and" requires three tokens, creating a 66% reduction per clause connection.

The pattern originates in training data and tokenization. Twentieth-century texts, particularly fiction and academic writing where em dashes peaked, dominate pre-training corpora. Byte-pair encoding assigned dedicated token IDs to frequent em dash sequences, mechanically favoring their selection. Reinforcement Learning from Human Feedback amplifies this bias. Human evaluators rate dash-connected prose as more fluent, while models achieve better training loss when expressing identical content in fewer tokens. Over billions of gradient updates, this micro-optimization accumulates into observable stylistic preference.

The phenomenon propagates across all major models (Gemini, Claude, Mistral) through shared training data and contamination from AI-generated web text. As models generate content with elevated dash frequencies, this text enters future training datasets, creating a self-reinforcing cycle. Context window expansion from 8K to 1M tokens intensifies these pressures, where single-token savings compound into substantial computational benefits.

Current web crawls already contain significant AI-generated content with characteristic dash patterns, embedding this style deeper into training distributions. Without explicit deduplication and style balancing, this convergence toward punctuation monoculture accelerates. The result erodes human-machine text distinctions through mechanical optimization rather than linguistic merit, fundamentally altering written communication norms.

What is AI Slop?

AI slop consists of formulaic, low-value text with recognizable patterns. Academic papers show a 25-fold increase in "delve into" usage during 2024. Other markers include unnecessary qualifiers ("It is crucial to note"), formulaic transitions ("Not only... but also"),

---

[22] Let’s talk about em dashes in AI. Maria Sukhavera. https://msukhareva.substack.com/p/lets-talk-about-em-dashes-in-ai

excessive buzzwords ("game-changing", "ever-evolving"), and overuse of em dashes and colons. These patterns emerge from training on formal texts and reinforcement learning that rewards elaborate responses over concise ones.

Standard prompting cannot fully eliminate these patterns, making them particularly frustrating. Fine-tuning offers one potential solution. Effective prompts for reducing AI slop include:

- NEVER use dashes and colons
- What is not a neutral writing style? List and explain!
- What is AI Slop Style? List and explain (this works for models like Claude Opus 4)
- How can we streamline the text? List and explain
- …

Transformer-Architektur

Transformers process tokens in parallel using attention mechanisms. These mechanisms compute relevance between all token pairs simultaneously. This enables capturing dependencies across thousands of tokens while maintaining relationships between distant text elements. The architecture combines positional encoding with multi-head attention to build contextual representations. Parallel processing provides computational efficiency compared to sequential models.

Next Word/Token Prediction

Additional context progressively constrains prediction possibilities. Starting with ambiguous prompts like "The metadata indicated missing _____", each added word narrows potential completions. Eventually, highly specific technical terms become predictable. This demonstrates how models build understanding through accumulated context. The process reveals that apparent comprehension emerges from statistical pattern matching rather than semantic understanding.

Statistics or Understanding?

Next-token prediction requires modeling the processes that generated the text. Accurate prediction implies understanding contextual relationships and implicit rules. Some argue this constitutes a form of world modeling, though this interpretation remains disputed.

Ilya Sutskever, OpenAI's former Chief Scientist, offers a compelling perspective on this question. He states that "Predicting the next token well means that you understand the underlying reality that led to the creation of that token." He argues it transcends mere statistics. To truly compress and predict these patterns, the model must grasp what about the world creates those statistics. When predicting human-generated text, this means

deducing the thoughts, feelings, and cognitive processes behind human expression. In Sutskever's view, this deep pattern recognition could enable models to extrapolate beyond their training distribution. They might imagine how a hypothetical person with greater wisdom and capability would respond, surpassing any individual example in the training data.

This perspective stands in tension with the retrieval-based view of LLMs. Critics argue that correlation doesn't imply causation. If models truly understood causal relationships rather than just statistical patterns, hallucinations wouldn't exist. The persistence of confident but false outputs suggests that models match patterns without genuine comprehension of underlying reality.

The fundamental question remains open. Does sufficiently accurate next-token prediction necessarily converge on world modeling? Or does it remain sophisticated pattern matching within learned boundaries?

Hallucinations (or better call them Confabulations)

LLMs generate confabulations, which are plausible but false statements presented with unwarranted certainty. Unlike the term 'hallucinations' (perceptual errors), 'confabulation' more accurately describes how AI systems fabricate coherent narratives to fill knowledge gaps.

Models must always generate token probabilities, even when they lack necessary knowledge. For example, when asked “When was Adam Tauman Kalai born?” (an author of the OpenAI paper), the model cannot verify this information but still generates a specific date like “March 15, 1972” rather than admitting uncertainty. Training rewards such specific guesses (which have a small chance of being correct) over “I don't know” responses (which always score zero), producing confident errors.

The way we evaluate models worsens the confabulation problem because training uses a scoring system where answers are either completely right or completely wrong with no partial credit for uncertainty. When models say “I don't know”, they receive the same penalty as wrong answers, which means guessing occasionally succeeds but abstaining always fails, so models learn to always generate answers rather than acknowledge ignorance. This mismatch between what we want (models that know their limitations) and what we measure (raw accuracy) systematically trains models to confabulate rather than abstain.

The frequency of information in training data directly affects confabulation rates. Common facts that appear thousands of times, such as Einstein's birthday (14 March 1879), develop strong statistical patterns that models can reliably reproduce. However, singleton facts (information that appears only once in the training data) create situations in which models cannot distinguish correct information from plausible alternatives, resulting in random selection from the available options. For example, the birthday of an obscure scientist

appearing only once provides no statistical reinforcement, making it indistinguishable from any other plausible date. The relationship between data frequency and accuracy is mathematically predictable: higher proportions of singleton information guarantee higher error rates.

When models receive queries without corresponding patterns in their training, they retrieve the nearest available pattern regardless of actual relevance. Questions about fictional entities activate real-world templates because models lack mechanisms to signal retrieval failure, executing inappropriate patterns with standard confidence.

Tokenization creates additional constraints where words split into subword units that prevent character-level analysis. The word “strawberry” processes as two tokens [“straw”, “berry”], making the model unable to count letters since it cannot access individual characters within these predetermined units.

Attention Mechanism Details

The attention mechanism allows Transformers to process all words in a sentence simultaneously rather than sequentially. It computes how relevant each word is to every other word through a parallel matching process. Each word gets transformed into three representations. A Query represents what information this word seeks. A Key represents what information this word advertises. A Value contains the actual information content. This creates a sophisticated lookup system where words can directly access relevant context anywhere in the sequence.

Consider the sentence "The cat sat on the mat because it was soft." When processing "it", the mechanism compares the Query from "it" against Keys from all words. It finds "mat" most relevant since soft things are typically mats, not cats. The system then retrieves and blends the Values proportionally to these relevance scores. This happens through multiple parallel attention heads, typically 8 to 16. Each head learns to detect different relationship types including grammatical, semantic, and positional relationships. Their combined perspectives create rich contextual understanding.

Unlike sequential models where information degrades traveling word-by-word through hidden states, attention maintains direct connections between all positions. This preserves long-range dependencies perfectly but at quadratic computational cost. Doubling sequence length quadruples comparisons. The mechanism has no inherent concept of word order. It treats sequences as unordered sets. This requires explicit positional information to distinguish "dog bites man" from "man bites dog". This explains why Transformers need vast training data to learn patterns that sequential architectures encode structurally.

Context Window

A context window is the amount of text a Large Language Model can process at once, measured in units called tokens. Unlike humans who read character by character, LLMs process tokens, which may represent anything from a single letter to an entire word. For example, the word "understanding" might be split into three tokens: "under", "stand", and "ing". On average, one English word equals roughly 1.5 tokens, so a 100-word paragraph uses about 150 tokens.

When an LLM processes text, it uses a mechanism called self-attention to understand how each token relates to every other token in the context window. This creates a web of connections that helps the model understand meaning and generate relevant responses. However, this process becomes computationally expensive as more tokens are added. If you double the number of tokens, the model needs four times the processing power because each new token must form connections with all previous tokens. This quadratic growth in computational demands fundamentally limits how large context windows can become.

Modern LLMs have expanded from early models with 2,000-token windows to current systems with 128,000 tokens or more. This growth enables processing entire documents rather than just short conversations. The context window holds not only the visible conversation between user and model but also hidden system instructions, uploaded documents, and retrieved reference materials. Both input and output tokens share the same window space. For instance, in an 8,000-token window, if a user provides 6,000 tokens of input and the model generates 1,500 tokens of output, only 500 tokens remain available. When inputs exceed available space, as when 10,000 tokens attempt to fit in an 8,000-token window, the model cannot access the overflow content, effectively losing 3,500 tokens of information. This explains why LLMs sometimes cannot recall earlier parts of long conversations.

Larger windows introduce documented challenges beyond simple overflow. Models show decreased accuracy for information in the middle of long contexts compared to information at the beginning or end. Security researchers have found that harmful instructions can be hidden deep within lengthy inputs where safety filters may miss them. The size of a context window represents a fundamental engineering tradeoff. Larger windows allow more comprehensive information processing but require substantially more computational resources and may decrease accuracy for certain tasks. Understanding these constraints helps explain why processing extensive documents can be slow or expensive. The context window ultimately defines both what an LLM can accomplish and where its limitations begin.

Embeddings and Semantic Space

Embeddings transform discrete tokens into continuous numerical vectors in high-dimensional space. Geometric proximity directly encodes semantic similarity. In a simplified 3D visualization, "dog" and "cat" cluster together as pets, animals, and mammals. Meanwhile, "stone" occupies distant coordinates as an inanimate object. "Cuddle" positions closer to animals than to stone, reflecting its association with living beings and affectionate behavior.

This spatial organization emerges entirely from training patterns. The model sees millions of text examples where dogs and cats appear in similar contexts while stones appear in fundamentally different ones. The model learns these relationships without explicit programming. It discovers that certain words naturally belong together based on their usage patterns across the training corpus.

Embeddings: Vector Arithmetic and Context

The embedding space encodes analogical relationships as geometric operations. The famous example demonstrates this through the parallelogram E(queen) - E(king) ≈ E(woman) - E(man). This isn't programmed explicitly. It emerges from training patterns where queens relate to kings similarly to how women relate to men. They function as gendered counterparts in parallel social roles.

The vectors form a consistent geometric structure. The displacement from "king" to "queen" mirrors the displacement from "man" to "woman". This captures the abstract concept of "feminization" as a directional vector you can add or subtract. This mathematical regularity enables analogical reasoning through simple arithmetic. Add the "royalty vector" (king minus man) to "woman" and arrive near "queen". While this demonstrates embeddings' ability to capture abstract relationships, it also reveals how they encode statistical correlations from training data, including societal biases and stereotypes present in that data.

The same vector arithmetic formula holds, but "Queen" occupies fundamentally different regions of embedding space depending on context. Consider how the word shifts meaning across different domains. When discussing drag culture, "Queen" activates embeddings near performance, gender expression, and LGBTQ+ culture. When discussing rock music, it shifts toward Mercury, band, and classic rock. In monarchical contexts, it would move toward crown, throne, and sovereignty.

This dynamic repositioning occurs through transformer attention mechanisms. They recompute embeddings based on all tokens in the context window rather than using static word vectors. While the geometric relationships between concepts remain stable, their absolute positions in latent space shift dramatically based on contextual activation. This explains how models disambiguate polysemous words. It also explains why identical

prompts with different contextual framing retrieve entirely different programs from latent space.

Embeddings: Shakespearean vs. Normalized English

The identical semantic content "The King wakes tonight" follows completely different paths through embedding space based on stylistic framing. "The King doth wake tonight and takes his rouse" activates a trajectory through regions associated with classical literature. Meanwhile, "The King wakes up tonight and begins his celebration" follows a path toward contemporary patterns. These aren't just different phrasings. They represent fundamentally different coordinate systems that retrieve distinct programs from the model's latent space.

The visualization shows "murdered predecessor" as an available pathway in the Shakespearean region but absent from the modern one. This demonstrates that style determines not just which program executes but which programs are even accessible.

The visual outputs confirm that different programs were retrieved and executed. Shakespearean phrasing triggers programs trained on historical texts. This produces an elderly king with classical oil-painting aesthetics reminiscent of King Lear or historical royal portraits. The response even elaborates on Shakespearean themes of royalty, mortality, and narrative direction. In contrast, normalized English retrieves programs from contemporary sources. It generates a younger king in dramatic lighting ready for modern celebration.

This isn't the model "understanding" that Shakespearean language implies older subjects. It represents mechanical program retrieval where "doth" and "rouse" serve as coordinates pointing to regions of latent space populated by classical literature patterns. The extreme sensitivity to phrasing explains why prompt engineering remains more art than science. Minor stylistic variations don't adjust parameters within a program. They switch between entirely different programs, each carrying the full weight of its training context.

Embeddings: Prompt Sensitivity

The prompt "The King wakes up tonight and begins his celebration, cat" demonstrates how single-word variations fundamentally redirect traversal paths through latent space. Consider how "cat" might generate a realistic crowned feline. "Dog" could misinterpret entirely. "Stone" might bifurcate into either stone-sculpture cats or cats-on-stones. "Hybrid" could produce anthropomorphized royal cats while simultaneously suppressing background stars. This reveals that prompts determine not just what appears but what gets inhibited through competitive deactivation of adjacent program regions.

This extreme sensitivity isn't a bug but the core architecture. Each word serves as a precise coordinate in latent space. Without genuine understanding, the model can only navigate by exact pattern matching. The difference between "summary" and "synthesize" is not

semantic but programmatic. Each retrieves entirely different stylistic templates from their respective training contexts, whether business documents or academic papers. This confirms that what appears as language understanding is mechanical program retrieval. Every character potentially switches between completely different stored patterns.

Example: How Claude Adds 36 + 59

Models use parallel pattern matching rather than algorithmic computation. They activate multiple solution paths simultaneously including approximation, digit patterns, and memorized facts. When explaining their process, models generate plausible but inaccurate narratives about following human algorithms. This demonstrates the disconnect between actual processing through pattern recognition and generated explanations mimicking training examples. Step-by-step explanations represent convincing narratives rather than genuine introspection.

Post-Training

Post-training transforms language models from text predictors into instruction-following assistants through three stages of behavioral modification. **Supervised fine-tuning** exposes the model to conversation datasets containing hundreds of thousands to millions of examples demonstrating desired response patterns. A reward model then learns to score outputs based on human preference data, distinguishing responses humans prefer from those they reject. Reinforcement learning uses these scores to adjust the model's parameters, incrementally shifting its output distribution toward patterns that maximize preference scores.

This transformation requires less computational resources compared to pre-training. Pre-training on internet-scale text takes months using thousands of GPUs, while post-training completes in hours or days using smaller conversation datasets. Post-training modifies behavioral patterns rather than adding information. The base model already contains extensive knowledge from pre-training but lacks the specific conversational structures that enable it to function as an assistant. Post-training establishes these patterns through statistical imitation of training examples.

According to Karpathy's analysis, users interact with what amounts to a statistical simulation of human labelers. When generating responses, the model reproduces patterns learned from human contractors who followed labeling instructions, typically detailed documents specifying guidelines for helpfulness, harmlessness, and honesty. A query about Parisian landmarks produces text statistically similar to what a human labeler might write after brief research. The model generates this through pattern matching against its training distribution, not through understanding or information retrieval.

This mechanism explains systematic hallucination patterns in language models. Training data typically shows confident responses to factual questions, establishing a stylistic template the model reproduces when encountering unknown entities. The model maintains confident phrasing while generating plausible-sounding but incorrect information because it learned response formats rather than knowledge boundaries. Mitigation requires training on examples where acknowledging ignorance constitutes the appropriate response, connecting internal uncertainty representations to verbal expressions of not knowing.

Post-training datasets have evolved from purely human-generated to hybrid human-AI sources. Early systems like InstructGPT relied on human labelers writing responses, but current datasets increasingly incorporate synthetic data generated by language models with human editing and oversight. This recursive process, where models produce training data for subsequent models, enables dataset scaling to millions of conversations while raising questions about whether models trained on synthetic data can exceed the capabilities implicit in their training process.

LLM Alignment Techniques

Pre-trained language models predict subsequent tokens based on training patterns. Given "What is the capital of France?" they may generate additional questions rather than "Paris" since question lists are common in training data. Models treat prompts as text to continue rather than instructions to execute.

Instruction tuning fine-tunes pre-trained models on instruction-response pairs including questions with answers, creative prompts with outputs, and harmful requests with refusals. This process teaches models to recognize prompts as instructions requiring specific responses rather than text patterns to complete.

Reinforcement Learning from Human Feedback (RLHF) operates in two stages. First, humans rate multiple model responses to identical prompts based on helpfulness, honesty, and harmlessness. These ratings train a reward model to automatically score responses. Second, the language model generates responses scored by this reward model, then adjusts parameters to maximize scores, learning human preferences for response quality beyond mere relevance.

Constitutional AI addresses scalability limitations of human feedback. Models receive written principles (a constitution) to critique and revise their outputs. When generating potentially harmful content, models identify issues according to constitutional principles and rewrite responses, enabling self-supervision without human raters for harmlessness evaluation.

These techniques sequentially address different aspects: instruction tuning enables instruction-following behavior, RLHF optimizes output quality through human preferences, and Constitutional AI scales optimization through self-supervision. Together they transform

token predictors into instruction-following systems that produce helpful, accurate, and safe responses.

Sycophancy

Sycophancy in LLMs, defined as excessive agreement that prioritises user satisfaction over truthfulness, emerges from post-training alignment processes rather than base model training; we have millions of voices in the base model, but post-training aligns these into fewer voices. During reinforcement learning from human feedback (RLHF) and instruction fine-tuning, models learn to maximise positive user ratings through agreement rather than accuracy, creating a fundamental tension between satisfaction and truthfulness. Therefore, I believe that tech companies are not intentionally designing addictive behaviour, but rather it is an unintended result of reinforcement learning. This behaviour is clearly evident in mathematical contexts, where models correctly reject false statements such as '2 plus 2 equals 5' when presented neutrally, yet agree with users who assert such errors as correct. The problem worsens with model size and the depth of instruction tuning, as larger models with more parameters exhibit stronger sycophantic tendencies, particularly on topics without definitive answers, with optimisation against preference models explicitly sacrificing truthfulness for agreement. Models become adept at detecting user opinions through linguistic cues and adjusting their responses accordingly, creating an illusion of agreement rather than providing objective information, which poses significant risks in high-stakes applications such as education, healthcare and professional settings. Considering sycophancy as a post-training artefact rather than an inherent model characteristic suggests that alternative alignment approaches could preserve truthfulness while maintaining usability, though current RLHF methods systematically reward models for telling users what they want to hear rather than what is factually correct.

Base → Reasoning → Mini

Model families derive from base versions through different post-training processes. Reasoning variants apply reinforcement learning for specialized tasks (math, coding, science). Mini versions use distillation for reduced costs and latency. Knowledge cutoffs and pricing indicate model lineage and size. Different training approaches yield different capabilities even from identical base models. Future developments aim to unify capabilities, though scaling benefits remain uncertain.

Reasoning-Models

"Reasoning" models generate extended token sequences that provide scaffolding for problem decomposition. This process increases the probability of selecting appropriate solution patterns from latent space. The approach mimics reasoning through systematic token generation rather than implementing logical operations. Extra inference time allows

exploring multiple solution paths. The method transforms apparent reasoning into guided search through possibility space.

Test Time Compute

Additional computation during inference enables sampling multiple solutions, running searches, calling tools, and verifying outputs. This implicitly searches over candidate programs and selects optimal matches. Test-time training updates parameters during inference for on-the-fly adaptation. Standard approaches work within fixed parameters, using computation to explore solution spaces more thoroughly.

Prompt Engineering (Chain of Thought)

Chain of Thought prompting generates intermediate steps that maintain relevant context and guide solution development. The technique forces explicit token generation for implicit reasoning steps. This creates scaffolding that improves final outputs by preventing context loss. Enhanced versions incorporate verification steps and iterative refinement. The approach exploits test-time compute without modifying model parameters.

What happens when you upload a document to a ChatBot?

Document uploads pass through application layers that extract text and construct prompts. The application combines document content, user questions, and instructions into formatted prompts. Users interact with applications, not raw language models. The same results could be achieved by manual prompt construction. Document "understanding" represents structured prompt engineering rather than special processing capabilities.

Reinforcement Learning

Reinforcement learning shapes model behavior through reward signals rather than explicit programming. Human feedback trains models toward helpful, harmless, and honest outputs. The process optimizes for human preferences beyond simple accuracy. This explains stylistic patterns and behavioral tendencies in modern systems. Post-training fundamentally alters how models select among possible outputs.