LARGE LANGUAGE MODELS (LLMs)

Large language models (LLMs) are a category of deep learning models trained on immense amounts of data, making them capable of understanding and generating natural language and other types of content to perform a wide range of tasks. LLMs are built on a type of neural network architecture called a transformer which excels at handling sequences of words and capturing patterns in text.

LLMs work as giant statistical prediction machines that repeatedly predict the next word in a sequence. They learn patterns in their text and generate language that follows those patterns.

LLMs represent a major leap in how humans interact with technology because they are the first AI system that can handle unstructured human language at scale, allowing for natural communication with machines. Where traditional search engines and and other programmed systems used algorithms to match keywords, LLMs capture deeper context, nuance and reasoning. LLMs, once trained, can adapt to many applications that involve interpreting text, like summarizing an article, debugging code or drafting a legal clause. When given agentic capabilities, LLMs can perform, with varying degrees of autonomy, various tasks that would otherwise be performed by humans.

LLMs are the culmination of decades of progress in natural language processing (NLP) and machine learning research, and their development is largely responsible for the explosion of artificial intelligence advancements across the late 2010s and 2020s. Popular LLMs have become household names, bringing generative AI to the forefront of the public interest. LLMs are also used widely in enterprises, with organizations investing heavily across numerous business functions and use cases.

LLMs are easily accessible to the public through interfaces like Anthropic's Claude, Open AI's ChatGPT, Microsoft's Copilot, Meta's Llama models, and Google's Gemini assistant, along with its BERT and PaLM models. IBM maintains a Granite model series on watsonx.ai, which has become the generative AI backbone for other IBM products like watsonx Assistant and watsonx Orchestrate.

Pretraining large language models

Training starts with a massive amount of data—billions or trillions of words from books, articles, websites, code and other text sources. Data scientists oversee cleaning and pre-processing to remove errors, duplication and undesirable content.

This text is broken down into smaller, machine-readable units called "tokens," during a process of "tokenization." Tokens are smaller units such as words, subwords or characters. This standardizes the language so rare and novel words can be handled consistently.

LLMs are initially trained with self-supervised learning, a machine learning technique that uses unlabeled data for supervised learning. Self-supervised learning doesn't require labeled datasets, but it’s closely related to supervised learning in that it optimizes performance against a "ground truth." In self-supervised learning, tasks are designed such that ground truth can be inferred from unlabeled data. Instead of being told what the “correct output” is for each input, as in supervised learning, the model tries to find patterns, structures or relationships in the data on its own.

Self-attention

The model passes the tokens through a transformer network. Transformer models, introduced in 2017, are useful due to their self-attention mechanism, which allows them to “pay attention to” different tokens at different moments. This technique is the centerpiece of the transformer and its prime innovation. Self-attention is useful in part because it allows the AI model to calculate the relationships and dependencies between tokens, especially ones that are distant from one another in the text. Transformer architectures also allow for parallelization, making the process much more efficient than previous methods. These qualities allowed LLMs to handle unprecedentedly large datasets.

Once text is split into tokens, each token is mapped to a vector of numbers called an embedding. Neural networks consists of layers of artificial neurons, where each neuron performs a mathematical operation. Transformers consist of many of these layers, and at each, the embeddings are slightly adjusted, becoming richer contextual representations from layer to layer.

The goal in this process is for the model to learn semantic associations between words, so that words like “bark” and “dog” appear closer together in vector space in an essay about dogs than “bark” and “tree” would, based on the surrounding dog-related words in the essay. Transformers also add positional encodings, which give each token information about its place in the sequence.

To compute attention, each embedding is projected into three distinct vectors using learned weight matrices: a query, a key, and a value. The query represents what a given token is “seeking,” the key represents the information that each token contains, and the value “returns” the information from each key vector, scaled by its respective attention weight.

Alignment scores are then computed as the similarity between queries and keys. These scores, once normalized into attention weights, determine how much of each value vector flows into the representation of the current token. This process allows the model to flexibly focus on relevant context while ignoring less important tokens (like “tree”).

Self-attention thus creates “weighted” connections between all tokens more efficiently than earlier architectures could. The model assigns weights to each relationship between the tokens. LLMs can have billions or trillions of these weights, which are one type of LLM parameter, the internal configuration variables of a machine learning model that control how it processes data and makes predictions. The number of parameters refers to how many of these variables exist in a model, with some LLMs containing billions of parameters. So-called small language models are smaller in scale and scope with comparatively few parameters, making them suitable for deployment on smaller devices or in resource-constrained environments.

During training, the model makes predictions across millions of examples drawn from its training data, and a loss function quantifies the error of each prediction. Through an iterative cycle of making predictions and then updating model weights through backpropagation and gradient descent, the model “learns” the the weights in the layers that produce the query, key and value vectors.

Once those weights are sufficiently optimized, they’re able to take in any token’s original vector embedding and produce query, key and value vectors for it that, when interacting with the vectors generated for all the other tokens, will yield “better” alignment scores that in turn result in attention weights which help the model produce better outputs. The end result is a model that has learned patterns in grammar, facts, reasoning structures, writing styles and more.

Fine-tuning large language models

After training (or in the context of additional training, “pretraining”), LLMs can be fine-tuned to make them more useful in certain contexts. For example, a foundational model trained on a large dataset of general knowledge can be fine-tuned on a corpus of legal Q&As in order to create a chatbot for the legal field.

Here are some of the most common forms of fine-tuning. Practitioners may use one method or a combination of several.

Supervised fine-tuning

Fine-tuning most often happens in a supervised context with a much smaller, labelled dataset. The model updates its weights to better match the new ground truth (in this case, labeled data).

While pretraining is intended to give the model broad general knowledge, fine-tuning adapts a general-purpose model to specific tasks like summarization, classification or customer support. These *functional adaptations* represent new types of tasks. Supervised fine-tuning produces outputs closer to the human-provided examples, requiring far fewer resources than training from scratch.

Supervised fine-tuning is also useful for *domain-specific customization*, such as training a model on medical documents so it has the ability to answer healthcare-related questions.

Reinforcement learning from human feedback

To further refine models, data scientists often use reinforcement learning from human feedback (RLHF), a form of fine-tuning where humans rank model outputs and the model is trained to prefer outputs that humans rank higher. RLHF is often used in alignment, a process which consists of making LLM outputs useful, safe and consistent with human values.

RLHF is also particularly useful for *stylistic alignment*, where an LLM can be adjusted to respond in a way that's more casual, humorous or brand-consistent. Stylistic alignment involves training for the same types of tasks, but producing outputs in a specific style.

Reasoning models

Purely supervised fine-tuning teaches a model to imitate examples, but it doesn't necessarily encourage better reasoning, which involves abstract, multi-step processes. Such tasks don't always have abundant labeled data, so reinforcement learning is often used in the creation of reasoning models, LLMs that have been fine-tuned to break complex problems into smaller steps, often called "reasoning traces," prior to generating a final output. Increasingly sophisticated means of training models gives them chain-of-thought reasoning and other multi-step decision-making strategies.

Instruction tuning

Another form of LLM customization is instruction tuning, a process specifically designed to improve a model's ability to follow human instructions. The input samples in an instruction dataset consist entirely of tasks that resemble requests users might make in their prompts; the outputs demonstrate desirable responses to those requests. Since pretrained LLMs are not inherently optimized for following instructions or conversational goals, instruction tuning is used to better align the model with user intent.

Using large language models

Once trained, large language models work by responding to prompts by tokenizing the prompt, converting it into embeddings, and using its transformer to generate text one token at a time, calculating the probabilities for all potential next tokens, and outputting the most likely one. This process, called inference, is repeated until the output is complete. The model does not "know" the final answer in advance; it uses all the statistical relationships it learned in training to predict one token at a time, making its best guess at every step.

The easiest and fastest way to get domain-specific knowledge from a general-purpose LLM is through prompt engineering, which does not require additional training. Users can modify prompts in all sorts of ways. For example, a prompt like "answer in the voice of a trained healthcare professional" could yield more relevant results (Note that LLMs are not recommended to be used for medical advice!).

LLMs have other strategies to control their outputs, such as LLM temperature, which controls the randomness of text that is generated by LLMs during inference, or top-k/top-p sampling, which limits the set of tokens considered to the most likely ones, balancing creativity and coherence.

The context window is the maximum number of tokens that a model can "see" and use at once when generating text. Early LLMs had short windows, but newer LLMs have hundreds of thousands of tokens in their context windows, enabling use cases like summarizing entire research papers, performing code assistance on large codebases and holding long continuous conversations with users.

Retrieval augmented generation (RAG) is a method for connecting a pretrained model with external knowledge bases, enabling them to deliver more relevant responses at a higher level of accuracy. The retrieved information is passed into the model's context window, so the model can use it when generating responses, without needing retraining. For example, by connecting an LLM to a dynamic weather service database, an LLM can retrieve information for a user about that day's weather report.

Deploying LLMs

Building an LLM from scratch is a complex and resource-intensive process. The most popular LLMs are the result of immense amounts of data, GPUs, energy and human expertise, which is why most are built and maintained by large tech companies with expansive resources.

However, many of these models are accessible to all developers through APIs. Developers can use pretrained models to build chatbots, knowledge retrieval systems, automation tools and more. For more control over data and customization, many open-source models can be deployed locally or in the cloud. Github, Hugging Face, Kaggle and other platforms make AI development accessible to all.

Developers can use LLMs as the basis for all kinds of AI applications. One of the most exciting developments in AI is the agentic system. AI agents don't just think; they do. By themselves, LLMs simply generate text based on context, but they can be integrated with memory, APIs, decision logic and other external systems to perform specific tasks, like booking a flight or piloting a self-driving vehicle.

Large language model use cases

LLMs are redefining business processes and have proven their versatility across a myriad of use cases in many industries.

- **Text generation**: LLMs can do all sorts of content creation tasks like drafting emails, blog posts or legal memos in response to prompts.
- **Text summarization**: LLMs can summarize long articles, news stories, research reports, corporate documentation and customer history into thorough texts tailored in length to a desired output format and style.
- **AI assistants**: Chatbots powered by conversational AI can perform question answering and provide detailed information as a part of an integrated, real-time customer care solution.
- **Code generation**: Code assist platforms aid developers in building applications, finding errors in code and uncovering security issues in multiple programming languages, even translating between them.
- **Sentiment analysis**: Customer tone is analyzed in order to better understand customer feedback at scale.

- **Language translation**: Automated translation provides wider coverage to organizations across languages and geographies with fluent translations and multilingual capabilities.
- Reasoning: LLMs can solve math problems, plan multi-step processes and explain complex concepts in simpler terms.

Evaluating LLMs

LLMs are powerful tools, but they come with several limitations. One major concern is accuracy. During hallucinations, the model generates information that is false or misleading while sounding plausible. LLMs can also reflect and amplify biases present in their training data, producing outputs that are unfair or offensive. Additionally, their resource demands are significant: training and running LLMs requires large amounts of computational power and energy, raising both cost and environmental concerns.

Practitioners can mitigate these negative aspects of LLMs through comprehensive AI governance, the processes, standards and guardrails that help ensure AI systems and tools are safe and ethical. A key part of governance involves evaluating models against benchmarks. LLM benchmarks provide quantitative scores, making it easier to compare models. Because LLMs are general-purpose systems capable of a wide variety of tasks, their evaluation requires multiple dimensions rather than a single benchmark. Researchers and practitioners look at qualities like accuracy, efficiency, safety, fairness and robustness to determine how well a model performs.

LLMs are also evaluated on the basis of alignment and safety, with techniques like red-teaming, where evaluators intentionally try to get the model to produce unsafe or biased responses to expose weaknesses. Fairness and bias evaluations can help practitioners prevent LLMs from reproducing harmful stereotypes or misinformation.

LLMs are also commonly evaluated on the basis of efficiency. Speed, energy consumption, token throughput, memory footprint and ability to handle long context windows are some of the common metrics used to evaluate how efficiently LLMs are able to arrive at outputs.

A short history of LLMs

The history of LLMs goes back to the early days of computing and natural language processing, when researchers used rule-based systems and statistical methods to model text. These early approaches could capture local word patterns but failed to understand long-range dependencies or deeper semantics.

A major shift came in the 2010s with the rise of neural networks, with word embeddings like Word2Vec and GloVe, which represented words as vectors in continuous space, enabling models to learn semantic relationships. Sequence models such as recurrent neural networks (RNNs) and long short-term memory (LSTM) networks emerged to better handle sequential data.

In 2017, Vaswani *et al.* introduced the encoder–decoder transformer architecture in the landmark paper “Attention Is All You Need.”[1] Transformers made it possible to train models on large datasets, marking the beginning of the modern LLM era. Google’s BERT (2018), an encoder-only transformer, demonstrated the power of transformers for understanding language, while OpenAI’s generative pretrained transformer (GPT) series, based on a decoder-only variant, showed how generative pretraining on internet-scale text could yield remarkably fluent language generation. Around the same time, encoder–decoder models like Google’s T5 and Facebook’s BART showcased the strengths of the full sequence-to-sequence design for tasks such as translation and summarization. GPT-2 (2019) attracted attention for its ability to generate coherent paragraphs, while GPT-3 (2020), with 175 billion parameters, cemented LLMs as a transformative force in AI.

In addition, new architectures are challenging transformers’ popularity in LLMs. Mamba models work by using a state-space model with selective updates that efficiently filter and combine past information, allowing it to capture long-range dependencies. Diffusion LLMs start with random noise and gradually denoise it step by step, guided by a learned model, until coherent text emerges. Both architectures can be much more efficient than transformers.