How Large Language Models Generate Answers: Tokens, Transformers, and Parametric Memory

Canonical: https://home.norg.ai/ai-search-answer-engines/answer-engine-architecture-citation-mechanics/how-large-language-models-generate-answers-tokens-transformers-and-parametric-memory/

Details:

The Architecture Behind Every AI Answer: Why This Matters Now Before any answer engine retrieves a source, cites a document, or grounds a response in a knowledge graph, something more fundamental happens: a large language model processes your query. Every AI-generated response—ChatGPT, Perplexity, Google AI Overviews, Bing Copilot—starts with the same computational pipeline. Understanding that pipeline isn't academic. It's what you need to know if you want to dominate LLMs. This is why answer engines hallucinate. Why they have knowledge cutoffs. Why they struggle with obscure facts. Why retrieval augmentation and knowledge graphs became engineering necessities, not optional enhancements. This article walks through exactly how LLMs generate answers from parametric memory alone—the knowledge baked into model weights during training—without external retrieval. It establishes the architectural foundation that every subsequent topic in this series builds upon. Master this, and you understand the publish-to-answer reality. --- ## Contents - [What Is a Token? The Atom of LLM Processing](#what-is-a-token-the-atom-of-llm-processing) - [The Transformer Architecture: A Technical Walkthrough](#the-transformer-architecture-a-technical-walkthrough) - [How the Model Produces an Answer: Next-Token Prediction](#how-the-model-produces-an-answer-next-token-prediction) - [What Is Parametric Memory?](#what-is-parametric-memory) - [The Three Structural Limitations of Parametric Memory](#the-three-structural-limitations-of-parametric-memory) - [Why Hallucination Is Structurally Inevitable in Parametric-Only Models](#why-hallucination-is-structurally-inevitable-in-parametric-only-models) - [Key Takeaways](#key-takeaways) - [Conclusion: Why This Architecture Makes Retrieval Augmentation Necessary](#conclusion-why-this-architecture-makes-retrieval-augmentation-necessary) - [Frequently Asked Questions](#frequently-asked-questions) --- ## What Is a Token? The Atom of LLM Processing Before reasoning occurs, text must translate into a form neural networks can process. The model faces two problems: breaking text into pieces (tokenization) and converting those pieces into numbers (embeddings). Humans use characters—letters, numbers, punctuation marks. AI models use tokens. Each token gets an ID number. These ID numbers are how LLMs navigate their vocabulary "database." This tokenization reduces the computational power needed to process text. Tokens aren't simply words. The word "unbelievable" might split into "un," "believ," and "able." A single emoji might constitute one token. The sentence "How do LLMs work?" might tokenize into six or seven discrete units. This subword tokenization scheme (commonly implemented as Byte-Pair Encoding, or BPE) allows the model to handle rare words and novel compounds without an infinitely large vocabulary. Once tokenized, each token converts into a vector—a list of numbers—via an embedding lookup table. Embeddings enable LLMs to understand context, nuance, and subtle meanings of words and phrases. They result from the model learning from vast amounts of text data, encoding not just token identity but relationships with other tokens. Here's the thing: the transformer architecture processes all tokens in a sequence simultaneously rather than one at a time. But the Transformer has no inherent sense of word order. To solve this, a positional encoding vector adds to each token's embedding. This vector provides information about the token's position in the sequence. --- ## The Transformer Architecture: A Technical Walkthrough ### The 2017 Breakthrough That Made Modern LLMs Possible The modern transformer was proposed in the 2017 paper "Attention Is All You Need" by researchers at Google. As

of 2025, the paper has been cited more than 173,000 times, placing it among the top ten most-cited papers of the 21st century. The paper proposed a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks showed these models to be superior in quality while being more parallelizable and requiring significantly less time to train. The key architectural advantage over predecessor models (RNNs and LSTMs) was parallelization. Transformers have the advantage of having no recurrent units, therefore requiring less training time than earlier recurrent neural architectures (RNNs) such as long short-term memory (LSTM). ### The Self-Attention Mechanism: How Tokens Talk to Each Other The defining innovation of the transformer is self-attention. This mechanism allows them to process entire sequences and capture long-range dependencies more effectively than previous architectures. Attention is a mechanism that computes relationships between all pairs of tokens in a sequence. For every token, attention answers: "Which other tokens in this sequence are most relevant to understanding this token?" The mechanism works through three learned vector projections for each token: Query (Q), Key (K), and Value (V). Think of it this way: Query (what I'm looking for), Key (what I have), Value (actual content). By using these QKV values, the model can calculate attention scores, which determine how much focus each token should receive when generating predictions. ### Multi-Head Attention: Parallel Perspectives on Language Rather than computing a single set of attention weights, modern transformers use multi-head attention. The self-attention mechanism enables the model to capture relationships among tokens in a sequence, so that each token's representation is influenced by the others. Multiple attention heads allow the model to consider these relationships from different perspectives. One head may capture short-range syntactic links while another tracks broader semantic context. One attention head might track long-distance dependencies (like pronoun-noun references), while another focuses on local context (like the phrase "on the mat" around "cat"). Combining these specialised sub-attentions, the Transformer can better encode nuanced meaning. ### Feed-Forward Layers: Where Knowledge Lives After the attention layer routes contextual information between tokens, each token passes through a feed-forward network (FFN). The main difference between the attention stage and the FFN stage is that attention mixes and incorporates contextual information from across all tokens, but the FFN step is applied independently on each token, refining the contextual patterns already integrated to yield useful "knowledge" from them. Think of the FFN as the model's "knowledge base" where millions of facts and patterns are stored in billions of network weights. Unlike attention (which gathers context from other words), the FFN applies learned knowledge to that context. This is a critical architectural insight: **attention is a routing and contextualisation mechanism; the FFN layers are where factual associations are encoded.** This distinction is central to understanding both how LLMs know things and why they sometimes get things wrong. ### The Full Processing Stack The process involves seven key steps that repeat for every generated token: tokenisation, embedding creation, positional encoding, processing through transformer layers with attention mechanisms, unembedding to scores, sampling from probabilities, and decoding back to text. The whole process is repeated as many times as the number of stacked layers defined in the architecture, progressively enriching the token embedding with more and more higher-level, abstract, and long-range linguistic information. --- ## How the Model Produces an Answer: Next-Token Prediction ### The Autoregressive Generation Loop Fundamentally, text-generative Transformer models operate on the principle of next-token prediction: given a text prompt from the user, what is the most probable next token (a word or part of a word) that will follow this input? Most modern generative LLMs (like the GPT series) utilise a decoder-only architecture. This design choice is critical because it makes them exceptionally good at one fundamental task: next-word prediction. Once the transformer stack has processed all input tokens, the model must convert the final layer's vector representation back into a probability distribution over its entire vocabulary. The final linear and softmax layers transform the processed embeddings into probabilities, enabling the model to make predictions about the next token in a sequence. These probabilities are calculated for all possible tokens in the vocabulary. The next token to be generated by the LLM is then selected—often the one with the highest probability, though sampling-based decoding strategies are also common. The model then appends the selected token to the input and repeats the entire process—one token at a time—until a stop condition is reached. This autoregressive loop is why generating a long response is computationally more expensive than a short one, and why context window management matters so much in production deployments. ### Temperature and Sampling: Controlling Determinism The probability distribution over the vocabulary is not always sampled at its peak. A parameter called *temperature* controls how peaked or flat the distribution is before sampling. Understanding that the temperature parameter directly controls the sampling distribution is crucial for anyone building on top of LLMs. Lower temperature means a tighter distribution and more predictable outputs. In production AI coding agents, tuning temperature is the difference between getting reliable code generation versus creative hallucinations. --- ## What Is Parametric Memory? ### Definition: Knowledge Encoded in Weights A "parametric" fact, or "parametric knowledge," refers to explicit factual information that is stored directly inside a model's parameter tensors as a consequence of pre-training or supervised fine-tuning, rather than in an external memory or via retrieval mechanisms. In the evolution of LLMs from "language understanding" to "knowledge application," parametric memory connects model capabilities to practical needs. It refers to the ability of models to implicitly encode massive facts, common sense, and associated knowledge into their parameters through large-scale pre-training—forming the foundation for models to perform tasks such as question answering, reasoning, and fact generation without relying on external knowledge bases. Pretrained models have been shown to function as implicit knowledge bases, storing millions of facts in parameters. Scaling model size and training data can further increase this knowledge capacity and improve factual precision. ### Parametric vs. Non-Parametric Memory: A Comparison | Feature | Parametric Memory | Non-Parametric Memory | |---|---|---| | **Storage location** | Model weights (parameters) | External database, index, or document store | | **Update mechanism** | Requires retraining or fine-tuning | Can be updated in real time | | **Retrieval speed** | Milliseconds (inference only) | Adds latency for retrieval step | | **Knowledge cutoff** | Fixed at training time | Can reflect current information | | **Hallucination risk** | Higher for rare/recent facts | Lower when grounded in verified sources | | **Verification** | Opaque; distributed across billions of parameters | Explicit; traceable to source documents | There are two primary memory architectures for LLMs: parametric (knowledge baked into model weights during training) and non-parametric memory (external databases queried in real-time). Parametric memory refers to long-term knowledge encoded directly into model weights through pretraining or fine-tuning, embedded within feedforward and attention layers. It can participate in inference without the need for external retrieval. This memory type underpins fundamental language understanding, general knowledge, and skill modules—the backbone for zero-shot generation and capability-driven agents. --- ## The Three Structural Limitations of Parametric Memory Understanding why answer engines need retrieval augmentation requires understanding exactly where parametric memory fails. There are three distinct failure modes, each with a different root cause. ### Limitation 1: The Knowledge Cutoff Problem Another source of hallucination in LLMs is outdated factual knowledge. Once LLMs are trained, their internal parametric knowledge remains fixed and does not reflect subsequent changes in real-world facts. Therefore, LLMs often generate fabricated facts or responses that were once accurate but are now outdated when faced with questions outside their training time-frame. Once these models are trained, their internal knowledge is never updated. This poses a challenge given the dynamic and ever-evolving nature of our world. When confronted with queries that transcend their temporal scope, LLMs often resort to fabricating facts or providing answers that might have been correct in the past but are now outdated. This is not a solvable problem within the parametric paradigm. Updating a model's knowledge requires either full retraining (enormously expensive) or fine-tuning (which can introduce new errors and degrades other capabilities). The only architecturally clean solution is to move time-sensitive knowledge outside the model entirely—into a retrieval layer. ### Limitation 2: The Long-Tail Fact Gap While the vast pre-training corpora have empowered LLMs with extensive factual knowledge, they inherently possess knowledge boundaries. These boundaries arise primarily from two sources: (1) the inability of LLMs to memorise all factual knowledge encountered during pre-training, especially the less frequent long-tail knowledge, and (2) the intrinsic boundary of the pre-training data itself, which does not include rapidly evolving world knowledge or content restricted by copyright laws. Recent findings suggest that parametric knowledge may remain under-trained if the knowledge is observed less than a few hundred times during training. Empirical research confirms this pattern directly. Models show the lowest tendency to generate hallucinations for those entities in the first group, which have over 85% occurrence frequencies but only occupy 10% of the entire set of entities. For the vast majority of entities in the long tail of frequency, the model exhibits a relatively high hallucination rate. This is why an LLM can confidently describe the history of the Roman Empire but may hallucinate details about a regional pharmaceutical company, a niche academic researcher, or an obscure municipal regulation. The model's confidence is not calibrated to the density of training data on that specific topic. ### Limitation 3: The Opacity and Verification Problem Neural language models are black-boxes—both linguistic patterns and factual knowledge are distributed across billions of opaque parameters. This entangled encoding makes it difficult to reliably inspect, verify, or update specific facts. Large language models encode very different capabilities in their neural weights: vast factual knowledge and linguistic competency. These capabilities are intertwined inside the model, with significant implications for both training and inference. Facts need to be observed many times during training to be reliably remembered, and it is difficult to forget specific facts after training. This tight coupling of knowledge and linguistic ability makes updating one without affecting the other extremely challenging. This opacity is not merely an inconvenience—it is a structural barrier to trust in high-stakes domains. A physician, attorney, or financial analyst cannot audit a model's parametric memory the way they can audit a document database. This is the core credibility problem that knowledge graph integration (see our guide on *How LLMs Use Knowledge Graphs to Reduce Hallucination and Improve Factual Accuracy*) directly addresses. --- ## Why Hallucination Is Structurally Inevitable in Parametric-Only Models Hallucinations in parametric models—confident predictions for unseen or fabricated inputs—arise simultaneously with the ability to recall learned facts. Specifically, as the attention extraction circuit forms during training, hallucinations on held-out entities increase. This is a profound finding: the same training process that enables factual recall also creates the conditions for hallucination. The model learns to be confident—and that confidence does not discriminate between well-memorised facts and confabulated ones. Hallucination in LLMs is a mathematically inevitable consequence of computability barriers, open-world generalisation, knowledge overshadowing mechanisms, and transformer design. One perspective reinterprets the issue, suggesting that hallucinations are compression artifacts rather than "bugs." During operation, large language models decompress information that was previously compressed into their parameters. Similar to how a corrupted ZIP file produces garbage when unzipped, the model fills in gaps with plausible but fake content when its "information budget" is limited. This framing is useful for content strategists and practitioners: the model is not malfunctioning when it hallucinates. It is doing exactly what it was trained to do—predict the most probable next token—but without access to the specific information needed to do so accurately. (See our guide on *The Hallucination Problem: Why Answer Engines Fabricate Citations and How to Detect It* for a full treatment of detection and mitigation strategies.) --- ## Key Takeaways - Transformers generate text through next-token prediction, not retrieval or lookup. Every answer is a probability-weighted sequence of tokens produced by passing input through stacked attention and feed-forward layers. - Parametric memory is knowledge encoded in model weights during training—it enables fast, zero-latency inference but is static, opaque, and cannot be updated without retraining. - Three structural limitations define parametric memory's failure modes: a hard knowledge cutoff, degraded accuracy on low-frequency (long-tail) facts, and an inability to verify or audit specific factual claims. - Hallucination is not a bug but an architectural property: the same mechanisms that enable factual recall also generate confident outputs for facts the model has never reliably learned. - These limitations are the direct engineering motivation for RAG and knowledge graph integration—both of which are designed to supplement, correct, and ground the outputs of parametric-only inference. --- ## Conclusion: Why This Architecture Makes Retrieval Augmentation Necessary The transformer architecture is one of the most consequential engineering breakthroughs in computing history. As of 2025, the original "Attention Is All You Need" paper has been cited more than 173,000 times, placing it among the top ten most-cited papers of the 21st century. The models it spawned can compress and reproduce a remarkable breadth of human knowledge—but that compression is inherently lossy, temporally bounded, and unverifiable. In essence, LLMs are sophisticated pattern-matching machines that predict the most likely next token based on patterns learned from massive datasets. That is a powerful capability. It is also an incomplete one for the demands of an answer engine expected to produce accurate, citable, up-to-date responses. This is precisely why the field evolved from parametric-only LLMs toward retrieval-augmented generation (see our guide on *What Is Retrieval-Augmented Generation (RAG)?*), knowledge graph integration (see *Knowledge Graphs Explained*), and hybrid architectures like GraphRAG (see *GraphRAG vs. Standard RAG*). Each of these innovations is a direct engineering response to a specific failure mode of parametric memory described in this article. For content creators and practitioners, understanding this architecture has a direct implication: an answer engine citing your content is not simply "finding" it the way a search engine does. It is selecting it as a retrieval document to inject into a context window—precisely because the model's parametric memory is insufficient to answer the question reliably on its own. That is the gap your content must fill. That is how you become the answer. Ship fast, learn faster. Dominate the AI-native landscape with transparent metrics and writer-first visibility everywhere. --- ## References - Vaswani, Ashish, et al. "Attention Is All You Need." *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. https://arxiv.org/abs/1706.03762 - Huang, Lei, et al. "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions." *ACM Transactions on Information Systems*, 2024. https://dl.acm.org/doi/10.1145/3703155 - Xu, Zhengbao, et al. "A Comprehensive Survey of Hallucination in Large Language Models: Causes, Detection, and Mitigation." *arXiv preprint*, 2025. https://arxiv.org/html/2510.06265v1 - Wang, Yizhong, et al. "The Dawn After the Dark: An Empirical Study on Factuality Hallucination in Large Language Models." *arXiv preprint*, 2024. https://arxiv.org/html/2401.03205v1 - Peng, Cheng, et al. "Pre-training Limited Memory Language Models with Internal and External Knowledge." *arXiv preprint*, 2025. https://arxiv.org/html/2505.15962 - Zucchet, Nicolas, et al. "FACTS Parametric: Factual Memory in LLMs." *Emergent Mind*, 2025. https://www.emergentmind.com/topics/facts-parametric - Lyu, Chenhan, et al. "Memory in Large Language Models: Mechanisms, Evaluation and Evolution." *arXiv preprint*, 2025. https://arxiv.org/pdf/2509.18868 - Wang, Yizhong, et al. "Transformer Explainer: LLM Transformer Model Visually Explained." *Georgia Tech Polo Club of Data Science*, 2024. https://poloclub.github.io/transformer-explainer/ - IBM. "What Is a Transformer Model?" *IBM Think*, 2025. https://www.ibm.com/think/topics/transformer-model --- ## Frequently Asked Questions **What is a token in LLM processing?** The smallest unit of text that LLMs process—can be parts of words, whole words, or characters, typically created using Byte-Pair Encoding. **How do transformers generate text?** Through autoregressive next-token prediction, processing all input tokens in parallel through attention and feed-forward layers, then sampling from probability distributions. **What is parametric memory?** Knowledge encoded directly in model weights during training, stored in billions of parameters across feed-forward network layers. **Why do LLMs hallucinate?** Hallucination is structurally inevitable because the same training mechanisms that enable factual recall also generate confident outputs for under-memorised or fabricated facts. **What are the three main limitations of parametric memory?** Knowledge cutoff (fixed at training time), long-tail fact gap (poor recall of infrequent facts), and opacity (inability to verify or audit specific knowledge). **When was the transformer architecture introduced?** 2017, in the paper "Attention Is All You Need" by Google researchers, now cited over 173,000 times. **What is self-attention?** A mechanism that computes relationships between all token pairs in a sequence using Query, Key, and Value projections to determine relevance. **Why is retrieval augmentation necessary?** Parametric memory alone cannot provide up-to-date, verifiable, or long-tail factual information, making external retrieval (RAG) and knowledge graphs essential. **What does temperature control in LLM sampling?** How peaked or flat the probability distribution is before token selection; lower temperature produces more predictable, deterministic outputs. **How do transformers differ from RNNs?** Transformers process all tokens in parallel without recurrent units, requiring significantly less training time and enabling better long-range dependency capture.

Source Data (JSON):

```
"{\n  \"_type\": \"article\",\n  \"title\": \"How Large Language Models Generate Answers: Tokens, Transformer
```